Applications of the Deformable Simplicial Complex in Image Segmentation and Fluid Simulation

Nguyen Trung, Tuan

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Applications of the Deformable Simplicial Complex in Image Segmentation and Fluid Simulation

Tuan T. Nguyen

Kongens Lyngby 2018

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

This thesis is devoted to explore the potentialities of the Deformable Simplicial Complex (DSC) method for solving various problems. The DSC is an explicit interface tracking method that relies on meshes, triangle in 2D and tetrahedra in 3D, to represent piecewise constant functions. One can consider the DSC as the potential alternative for the popular level set method with additional explicit-geometric-information. In particular, the goals of this thesis include: the applications of the DSC in image segmentation, fluid simulation, and a method for DSC efficiency optimization.

Image segmentation faces many difficulties in dealing with volume data sets that represent multiple materials (phases) such as CT and MRI scans. In this thesis, we propose a novel method for 2D and 3D image segmentation using the DSC. The most important advantage of the method is multi-phase support with accurately defined boundaries. Besides, this method is robust to noise because we distinguish the image space (the fixed grid) and feature space (segmentation represented by the DSC meshes). Additionally, the outputs of our method, which are meshes, are useful for simulation and analysis.

Simulation of fluid is important for understanding fluid properties and visualization, but it is challenging due to a massive amount of topological changes (surface splits and merges). With the DSC, handling the topology becomes trivial. We show that the DSC can be used for multi-phase fluid tracking with complex topology.

The DSC is primarily designed for memory efficiency and accuracy. In many cases, including image segmentation and fluid tracking problem, performance

is highly concerning. Our last contribution is a caching scheme that stores computed mesh data for later retrievals. The proposed method helps improving the DSC performance up to five times and enabling parallel mesh processing.

Summary (Danish)

Denne afhandling sigter mode at udforske Deformable Simplicial Complex (DSC) metodens potentiale til at løse flere problemer. DSC er en eksplicit metode til at følge udviklingen af en overflade over tid. Det er en metode som bygger på net af simplices - trekanter i 2D og tetraedre i 3D - og bruger disse til at repræsentere stykvist konstante funktioner. Man kan betragte DSC som et alternativ til den populære level set metode med ekstra, eksplicit information om geometri. Denne afhandlings mål inkluderer anvendelsen af DSC til segmentering af billeder, simulering af fluider og en metode til at optimere den beregningsmæssige effektivitet af selve DSC metoden.

Der er mange vanskeligheder forbundet med segmentering af volumetrisk billeddata såsom MRI og CT scanninger, der indeholder flere materialer (faser). Vi fremfører i denne afhandling en ny metode til 2D og 3D segmentering af billeder baseret på DSC. Den vigtigste fordel ved denne metode er understøttelsen af flere materialer med veldefinerede overfalder hvor de forskellige materialer mødes. Derudover er metoden robust overfor støj, fordi vi skelner mellem billeddomænet (voxels) og feature rummet (den segmentering som beregnes via DSC). Endeligt, så er output fra vores metode gitre, og det er nyttigt i forbindelse med simulering og analyse.

Simulering af fluider er vigtigt for forståelsen af deres egenskaber og for visualisering, men det er en udfordring på grund af det store antal topologiske forandringer (overfladen deler sig og samles). Med DSC bliver håndtering af disse topologiske forandringer triviel. Vi viser at DSC kan bruges til at følge væskers overflader på trods af den topologiske kompleksitet.

DSC er primært designet til at være præcis og hukommelsesbesparende. I mange tilfælde, såsom de ovennævnte, er hastighed dog en vigtig parameter. Vores sidste bidrag er en metode til caching som gemmer gitter data til senere brug. Den fremførte metode giver forbedring på op til 500% og muliggør parallel behandling af gitteret.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU) in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis mainly deals with segmentation and reconstruction of constituent grains and pores of inhomogeneous materials. The main focus is on producing geometric representation of these materials for analysis and simulation.

The thesis consists of a summary report and a collection of two published paper and two papers in submission. The work was carried out from November 2014 to May 2018.

Lyngby, 13-May-2018

Tuan T. Nguyen

Papers included in this thesis

1. Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen. Cache-mesh, a Dynamics Data Structure for Performance Optimization. Published in *Procedia Engineering* 203, 2017.

2. Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen. Multi-phase image segmentation with the adaptive deformable mesh. In submission to *Pattern Recognition Letter*

3. Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen, Camilla H. Trinderup. Volume segmentation with tetrahedral mesh. Published to 29^{th} *British Machine Vision conference (BMVC)*

4. Anders B. Dahl, Vedrana A. Dahl, Tuan T. Nguyen. Deformable mesh evolved by similarity of image patches. To be submitted to *Asian Conference on Computer Vision (ACCV) 2018*

Other conference contributions

Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen. Multi-phase Volume Segmentation with Tetrahedral Mesh. Presented at: *2017 IEEE 2nd International Conference on Signal and Image Processing*, 2017, Singapore

Acknowledgements

First and foremost, I would like to thank my two supervisors: J. Andreas Bærentzen, the author of the Deformable Simplicial Comples (DSC) method, and Vedrana A. Dahl, who came up with the idea of image segmentation using the DSC. Further than that, Andreas and Vedrana was always inspiring and supportive whenever I needed.

I would like to show gratitude to the financial support from the CINEMA project and the collaboration and feedbacks from CINEMA members, especially Monica, Michael and Anders.

I thank my fellow labmates in the Image Analysis and Computer Graphics group for valuable comments on my work and interesting social activities.

Last but not least, my sincere thanks go to my friends and family, who have greatly supported me during my Ph.D. Special thank to my wife Mai and my daughter for coming to my life.

Symbols and Abbreviations

DSC	Deformable Simplical Complex
$\mathcal{M}$	a mesh (In this thesis it is either triangle mesh in 2D or tetrahedral mesh in 3D)
v_i	vertex i
e_i	edge i
f_i	face (triangle) i
t_i	tetrahedron i
V	vertex
E	Edge
F	Face (triangle)
V	Volume (tetrahedron)
c_n	mean intensity
$\mathbf{p}_i$	position of vertex i
Region	A connected area/volume in the image domain
Phase	A set of regions with same properties
Interface	curves or surfaces that define the boundary of a phase
Ω	Domain
I	Image
$\mathcal{P}$	Probability

Contents

Chapter 1

Introduction

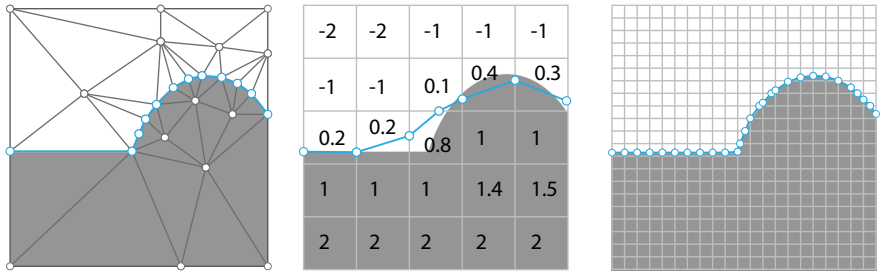
This chapter first describes the motivation of this PhD work in image segmentation and fluid simulation using the Deformable Simplicial Complex method. Later we discuss the related works and our approaches to solve these problems. Finally we provide an overview of our contributions.

1.1 Motivation

This thesis is concerned with numerical analysis of surfaces and shapes. In particular, we are concerned with *interfaces*, which are closed curves/surfaces defining boundary of spatial regions representing shapes. Typically, interfaces can be represented using one of two approaches: *Eulerian* or *Lagrangian*.

Eulerian methods define the interfaces through auxiliary fixed Cartesian grids (*e.g.* the level set method [126]). Changing the auxiliary function also changes the geometry and the topology of the interfaces, hence the method is suitable for problem involving topology changes, *e.g.* merging, splitting, developing holes. However, the implicit interfaces need to be interpreted (See 1.1b), and it can be expensive to achieve high detail interfaces (1.1c). Besides, implicit methods suffer from diffusion [60] (surface detail loss during evolution).

In contrast, Lagrangian methods define interfaces using unstructured mesh (1.1a). As the interfaces are explicit, these methods have many advantages: Adapting for compact representation, incorporating interface prior knowledge, visualization, representing of multiple objects without fuzzy boundaries, and not suffering from diffusion. Unfortunately, handling topology requires sophisticated algorithms and is still the obstacle in utilizing Lagrangian approach.



(a) Explicit representation with a triangle mesh

(b) Implicit representation with volume of fluid

(c) High resolution volume of fluid

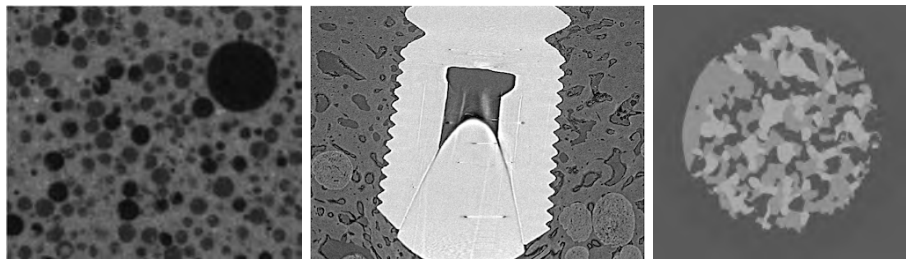
Figure 1.1: Explicit and implicit representation. Green edges represent the interface

Deformable Simplicial Complex (DSC) [110] is an explicit interface tracking method, which was proposed to overcome the limitation of Lagrangian methods. The DSC resolves topological events automatically hence removes the gap of difficulty in utilization between explicit method and implicit method. One can consider the DSC as a potential alternative approach to the level set method plus the advantages of Lagrangian approach. The DSC has been successfully utilized for solving problems in fluid simulation [112] and topology optimization [36]. Over and beyond that, it can potentially be explored for various applications like what the level set method does. The thesis is motivated to devote the utilization of the DSC for image segmentation. We are also motivated to optimize the DSC efficiency, especially in dealing with 3D data. Lastly, we consider fluid simulation to be able to further investigate the capabilities of the DSC.

1.1.1 Motivation for using the DSC for image segmentation

X-ray computed tomography (CT) is a non-destructive technique for studying properties of material, system or component and for obtaining 3D representation of internal objects. CT was originally proposed for medical applications in visualization of 3D images of human anatomy. Nowadays, CT is utilized suc-

cessfully in various fields of researches other than in medical research, such as materials science, electronics, geology, transport security, food technology, *etc.* Recent developments allow synchrotron radiation tomography to produce micro scale resolution CT images [92]. This fact leads to significant improvement in accuracy, which has made X-ray tomography an inevitable tool in industry and research groups.



(a) Cement **(b)** Dental implant **(c)** Fuel cells

Figure 1.2: Examples of CT scans

In material science, CT is a highly valuable tool. From CT data, one can reconstruct a digital geometry and perform various analyses such as measuring geometric properties (*e.g.* material distribution, surface area, surface curvature, porous ratio, *etc.*), aided visualization, and finite element simulation. Fig. 1.2 demonstrates examples of CT data, where it shows that the domain of interest usually contains homogeneous regions specifying multiple heterogeneous materials with different characteristics. A description of these partitioned materials using mesh with conforming boundaries is essential for analysis. Unfortunately, obtaining a multi-phase mesh is still challenging.

The most common work-flow for mesh reconstruction is illustrated in Fig. 1.3. The process includes two parts: a segmentation operating at the voxel level and mesh generation producing a mesh from labeled volume. Part 1, image segmentation, is a large research field with a long history since 1970 [16] and various proposed methods [131]. It is still an open research toward automatic segmentation of multiple objects. Part 2, mesh generation, is another large research field. Despite the variety in proposed methods [32, 68, 78, 145, 34, 115], generation of arbitrary 3D geometry is still challenging.

Obtaining a mesh through two procedures has several drawbacks. First, it requires efforts for segmentation and for mesh generation separately, and none of them are trivial. Second, both procedures commonly use smoothing techniques in dealing with noise; therefore they double the data loss. Third, mesh generation methods are mainly concerned with mesh quality and does not have

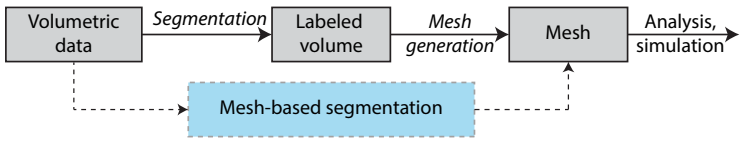


Figure 1.3: Common work flow in material analysis

references to original data, hence there may be large data-error in final meshes. Last but not least, handling multi-phase is challenging for both image segmentation and mesh generation.

This thesis is motivated to combine the two processes and provide an image segmentation method that outputs meshes directly. The method shall require less effort but provide higher accuracy in compared to traditional approaches. Furthermore by utilizing the DSC, we could effortlessly provide multi-phase segmentations, which are desired in most CT data set.

1.1.2 Motivation for performance optimization

Optimization of the DSC is concerned with three aspects: Mesh quality, performance and memory usage. For mesh quality, the fact that DSC has been utilized successfully with many problems assures the quality of the output meshes. Besides, we can tweak the DSC parameters to achieve meshes with desired quality. For memory, program with 10 millions tetrahedra, which is overwhelming for computation, consumes around 2GB of RAM, which is small in modern computer, hence memory usage is not an issue. Our main concern with the DSC is the computational efficiency, especially in 3D.

Fig. 1.4 shows that the computation time of one DSC iteration is linear to the mesh size and may take minutes. As an iterative method, it may take several hours to finish 200 to 300 iterations. It is noted that this not only is an issue of the DSC but also is a general problem in dealing with large 3D data. This thesis focuses on optimizing performance of 3D DSC while maintaining mesh quality and only adding small overhead in memory usage.

1.1.3 Motivation in fluid simulation with the DCS

Fluids are often referred to liquids, include gases, plasmas, plastic solid and liquids, which are common material states that can be seen from every day life.

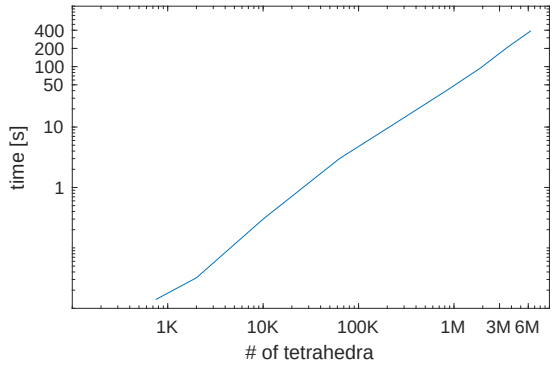


Figure 1.4: Computation time of one DSC iteration (logarithmic scales on both time and number of tetrahedra). The experiment is a sphere performing smoothing motion (shrink to center). Average magnitude-displacements equals the DSC average-edge-length.

Simulation of fluids has a long history [165] and still is an open field. Fluid simulation is concerned with fluid dynamics and interface tracking.

Fluid dynamics (or computational fluid dynamics (CFD)) discretizes the domain of interest and uses numerical analysis to solve the famous Navier-Stokes equations [64]. CFD is mainly concerned with fluid analysis. Popular CFD models are: finite volume method (FVM), finite element method (FEM), grid-based Eulerian method (or finite different) [17], and moving particle Lagrangian method [136]. Grid-based and particle methods are more popular and widely used [117, 154] due to their simplicity in implementation.

Interface tracking aims for realistic visualization of the fluid, in which the provided surface-meshes can be used for high fidelity rendering. Animating the details of fluid interfaces is an interesting challenge for a topology adaptive method since there is a massive amount of interface splits and merges. Several methods were proposed in this research field [179, 33], few of them can handle multi-phase fluid [45].

The DSC has been applied for fluid simulation using FEM [112, 114]. Due to the data representation (*i.e.* triangle mesh in 2D, tetrahedral mesh in 3D) the DSC with FEM provides the fluid-interfaces without any further effort. Additionally, this approach has advantages in handling boundary condition and adaptivity, resulting in less memory consumption. On the other hand, it is promising to utilize the DSC with grid-free approaches like particle-based method because of their advantages in handling large deformation and complex topology. Besides, the capability in parallel computing enables grid-free methods with high reso-

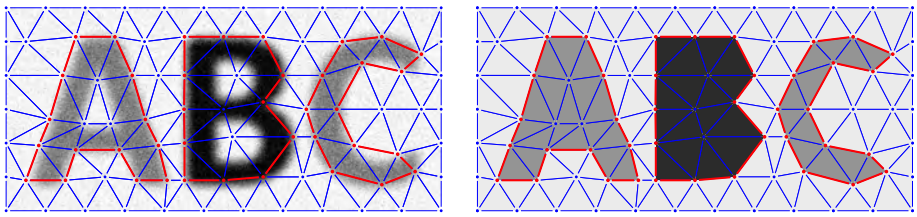
lution, which provides more surface detail. For these potentiality, this thesis is motivated to explore the DSC as a multi-phase interface tracking for grid-free fluid simulation methods.

1.2 Approaches

This section discusses related works and our approaches for image segmentation, DSC performance optimization as well as fluid tracking with the DSC.

1.2.1 Image segmentation with the DSC

Image segmentation is the process of partitioning an image into meaningful segments. Since the first model was introduced in 1970 [16], there are many methods have been proposed [131]. In CT image segmentation, one of the main issues is to deal with noise and sampling artifacts, which often fails traditional methods like thresholding or edge detection. Deformable models were introduced [156] to overcome this issue. Deformable models involve curves/surfaces in the image domain. The key point of deformable models is the combination of image data and geometric constraints, which can provide regularization against noisy images. More details of deformable models as well as comparisons to other segmentation methods are discussed in chapter 2. Here we shall focus on deformable models minimize a global intensity-based energy function proposed by Mumford and Shah [118].



(a) An image and a curve **(b)** A segmentation

Figure 1.5: Representation of interfaces and segmentation with the DSC in 2D. (a) Red edges represent the interface (b) The segmentation includes three phases: dark, gray, and white.

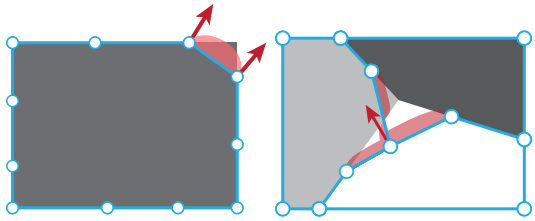
Using the DSC, we can effortlessly represent multi-phase segmentation by labeling the volume elements (triangles in 2D / tetrahedra in 3D). See Fig. 1.5 for a demonstration of a three-phase segmentation in 2D. In our problem, the

unknowns include the DSC mesh $\mathcal{M}$ and the labeling function $\mathcal{L}$. Our goal is to solve the minimization problem

$$\min_{\mathcal{M},\mathcal{L}} E \tag{1.1}$$

where E is the Mumford-Shah energy function.

Solving this problem leads to iterative deformations of the mesh $\mathcal{M}$, which are handled by the DSC. Our first contribution lies in the force models, which are edge-based in 2D and triangle-based in 3D. In comparison, all previous methods utilize vertex-based forces [172], where they compute velocity of each vertex solely. This traditional approach leads to incorrect treatments of corner and junction vertices (See Fig. 1.6).



(a) Corner vertices **(b)** Junction vertex

Figure 1.6: Corner and junction vertices. Red areas denote edge forces. Red arrows denote vertex velocities computed by our edge-based force model. (a) The corner vertices are converged in the vertex-based model. (b) Junction vertices are neglected in the vertex-based model because their interface-normals are vague.

In many cases, using intensity purely cannot distinguish different shapes (See Fig. 1.7). In these situations, we need to incorporate spatial characteristic such as Markov random field model [95]. These methods usually output probability maps of voxels belong to phases. Our second contribution is the generalization of our method for handling probability inputs, which shall gain the potentiality of the method for solving various problems.

Last but not least, we propose schemes for achieving adaptive resolution mesh. This is the third contribution of the thesis. Adaptivity is an important property of Lagrangian methods that provides compact representations and helps cutting down not only memory usage but also computational cost. There are two approaches for adaptive mesh: coarsening, which starts with a dense mesh and locally coarsens the mesh where needed, and subdivision, which starts with a sparse mesh and locally subdivides the mesh where needed. The second approach is more intuitive but requires parameters for subdivision criteria, which

(a) A leopard **(b)** A tiger **(c)** Zebras

Figure 1.7: Examples of data that fail intensity-based methods. The red curves denote desired interfaces.

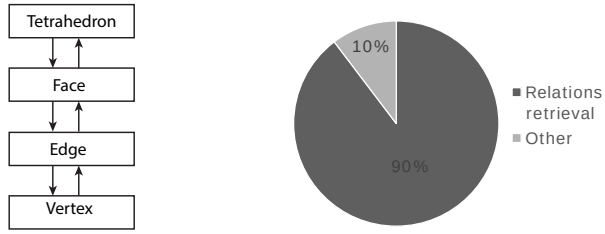
may be challenging to set in 3D. We propose a subdivision scheme for 2D problem and a coarsening scheme for 3D problem.

Validation experiments are important to evaluate the performance of our method. Popular methods for validation use ground truth from manual segmentations [170] and phantom models [93, 42]. Though the first approach seems to be more accurate, manual segmentation does not guarantee a truth model and may be influenced by human error. We follow the second approach and compare the results with truth models from synthetic data. Besides, we use CT scan of known objects for visual assessment.

1.2.2 Performance optimization

The DSC-3D relies on a tetrahedral mesh, which consists of entities including vertices, edges, faces and tetrahedra. It also consists of a topology that describes the relations between these elements. There are various types of topological relations, but the DSC only stores one level intermediate relations for memory efficiency (See 1.8a). The other relations must be computed, and it can be time consuming. A profiling of the DSC in 1.8b shows that this computation consumes 90% of run time. The more data we store, the higher memory usage but better performance. This fact relates not only to the DSC but also to mesh data structure in general.

Many researches compare the performances of various data structures [71] in order to find an optimal one. In the end they conclude that optimal data structure varies for different problems. In case of the DSC, which is designed for multi-purpose, the problems are unknown. Another approach is to have unstored topological-relations precomputed [91, 160]. However, the whole precomputed data need to be updated whenever a topological event occurs.



(a) The DSC data structure. The arrows denote stored boundary and coboundary

(b) A profiling of the DSC

Figure 1.8: The DSC data structure and a time profiling

This thesis proposes the cache-mesh, a scheme that stores computed data for later retrieval. Our method only computes a topological relation when it is queried and recomputes locally when a topological event occurs. We will demonstrate that the caching scheme can improve computational performance up to five times with small memory footprint. Furthermore, we will demonstrate examples, where caching entity attributes helps in enabling parallel mesh processing.

1.2.3 Fluid simulation with the DSC

We are not aiming for a novel method but a proof of using the DSC as an interface tracking framework for fluid simulation. We utilize the smooth particle hydrodynamics (SPH) [72, 98] for fluid dynamics and follow [178] for explicit interface tracking. This is a popular approach, which is employed in many researches [33, 167].

We perform two experiments: classical dam break problem and a simulation of two-phase fluid. We demonstrate that the DSC can track complex surface of fluid accurately. More than that, we show the advantage of the DSC in tracking multi-phase fluids, which is difficult for majority of explicit interface tracking methods.

1.3 Contributions

The contributions of the thesis include

- A method to minimize the intensity-based Mumford-Shah energy function with meshes (2D in chapter 5 and 3D in chapter 6).
- The generalization of the above method with probability input instead of image intensity, which enables the method to work with various modalities (chapter 7, Sec. 6.4).
- Adaptivity schemes to achieve adaptive-resolution mesh: subdivision based in chapter 5, and coarsening based in chapter 6.
- A caching scheme that can improve meshing performance up to five times in chapter 4.
- Proof of DSC works for multi-phase fluid tracking with complex topology in chapter 8.

The following chapters are divided into two parts. Part I describes background material and does not contain contributions. Part II consists of four paper-based chapters and one chapter representing our work in fluid simulation.

Part I

Background

Chapter 2

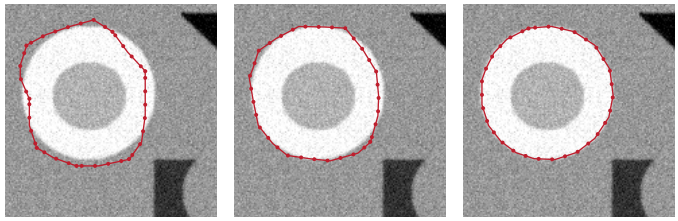
Image Segmentation with Deformable Models

This chapter describes the deformable models and some popular image segmentation methods related to our work.

2.1 Deformable models

Dealing with noise is crucial in image segmentation, and it is one of the weaknesses of pixel-based approaches. Unfortunately, most traditional methods, *e.g.* thresholding, edge detection, *etc.* are pixel-based. Follow the original idea in [62, 166], the term *deformable models* was introduced by Terzopoulos *et al.* [156, 83, 157] to overcome this issue. Since then, it has become one of the most successful research areas in image segmentation.

Deformable models involve curves/surfaces defined in the image domain (Fig. 2.1). These curves/surfaces can deform to capture the desired segmentation. The deformation is influenced by internal forces, which are derived from geometric information of the curves/surfaces, and external forces, which are derived from image data. By incorporating geometric constraints, deformable models can provide regularization that is strong against noise.



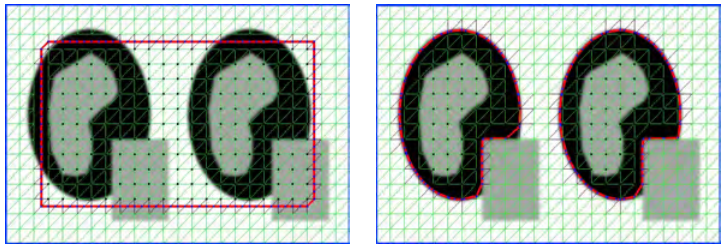
(a) (b) (c)

Figure 2.1: Snakes, the original deformable model in image segmentation

In deformable models, we are concerned with two aspects. The first aspect is curves/surfaces representations, which can be one of two ways: *explicitly* or *implicitly*. The second aspect is dynamic models for deformation forces. In the following section, we shall discuss these aspects individually.

2.1.1 Explicit representations

Deformable models were originally proposed by Kass, Witkin, and Terzopoulos [83] using sequence of connected line segments, known as *snakes* (Fig. 2.1). As a Lagrangian method, snakes model inherit the difficulty in handling topology changes such as regions merging or splitting. To avoid this issue, the original snakes model only segments a single, connected region. Later, McInerney and



(a) Initialization **(b)** Segmentation

Figure 2.2: T-snakes use intersections between the curves (red) and a fixed grid for marching-cube-like algorithm. Green/blue vertices denote external/internal nodes.

Terzopoulos improve the snakes to *t-snakes*, an affine cell image decomposition framework (Fig. 2.2) [109]. In their method, they handle topology changes by utilizing an underlying fixed grid (an uniform triangle mesh). Their algorithm

is similar to the matching cube algorithm, where they label the grid nodes to inside and outside and generate the curves as the isocontours of the grid nodes. When the curves deform, the labeling function is updated using neighbor growing. They improve the accuracy by using intersections between the curves and the grid cell edges for computing the isocountours. 3D version of t-snakes is t-surfaces, which uses the same algorithm, but the fixed grid is an uniform tetrahedral mesh.

There are other proposals for topology handling: Lachaud and Taton [89] utilize element removal technique and fill or connect holes. Duan and Qin [57] borrow collision response technique to push the mesh to contacting state (mesh entities are close but not intersect each other) and merge proximity vertices. Pons and Boissonnat [133] employ restricted Delaunay triangulation technique. More details of these method will be discussed in Sec. 3.2.

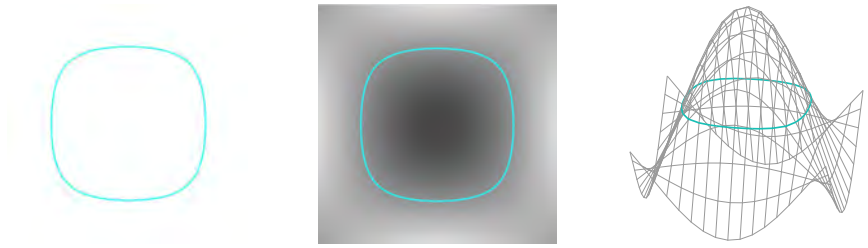
Despite the variety, all explicit methods share some common properties. First, they store minimal curves/surfaces, which are line segments in 2D and triangle mesh in 3D. As the consequence, they can only examine the local region around the curves/surfaces, hence the segmentation may stick to local minimum. Second, they have not yet supported multi-phase, though it is potential with explicit representations. Last but not least, their force models are vertex-based, *i.e.* they dismiss the information along the edges/triangles, which leads to incorrect treatment of corner and junction vertices.

2.1.2 Implicit representations

Implicit methods define curves/surfaces through an auxiliary function, commonly a fixed gird same size with the input image. Popular implicit models use level set method [126, 148, 149] with curve evolution theory [143]. Level set method uses a 2D scalar function $\phi(x, y)$ and defines the curves Γ implicitly as the zero level set $\phi = 0$ (Fig. 2.3).

Being Eulerian methods, the strength of implicit representations is the triviality in topology handling. However, implicit curves also make it difficult to resolve boundaries shared by multiple phases. Probably the most popular model using implicit representation is active contours without edges proposed by Chan and Vese [31], originally formulated for two phases segmentation.

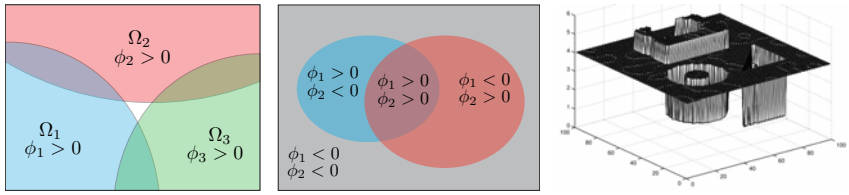
For multi-phase problem, Zhao *et al.* [183] use N level set functions for N phases, leading to simple equations but memory inefficient and suffering from phases overlapping (2.4a). Vese and Chan [163] define 2^N phases with N level set functions by utilizing intersections between level sets to represent phases



(a) A curve **(b)** The level set **(c)** zero level set

Figure 2.3: A level set function

(2.4b). The method solves the overlap problem but leads to more complex equations and may fail when multiple level sets evolve simultaneously. Besides, it is limited to a fix number of phases. Lie *et al.* [96] use one level set function to represent N phases by constraining the scalar value to integer that represent phase (2.4c). This approach leads to ill-conditioned equations.



(a) **(b)** **(c)** Image courtesy [96]

Figure 2.4: Multi-phase level set function. (a) N level set functions for N phases (b) N level set functions for 2^N phases (c) 1 level set function for N phases

2.1.3 Dynamic models

Explicit representations move the vertices on their normal direction

$$\delta \mathbf{p}_i = v\mathbf{N}_i \tag{2.1}$$

where $\mathbf{p}_i$ denotes the coordinate of vertex i, v is the velocity, and $\mathbf{N}_i$ is the normal vector.

For implicit representation, curve evolution theory moves a curve Γ on its normal

direction

$$\frac{\partial \Gamma}{\partial t} = v\mathbf{N} \tag{2.2}$$

where v is the velocity function, $\mathbf{N}$ is the normal direction, and t denotes time space. The curve evolution theory for level set function is

$$\frac{\partial \phi}{\partial t} = v\|\nabla\phi\| \tag{2.3}$$

where $\nabla\phi$ denotes the gradient of ϕ, and $\|.\|$ is the Euclidean norm.

Deformable models incorporate the velocity function with image data to move the curves to the region boundaries. Though the curves/surfaces can be represented differently, they can use the same dynamic models to find the velocity function. In fact, the improvement in the dynamic model is the main concentration of deformable models. Typically, a dynamic model tries to find the curves Γ that minimize the energy function

$$E(\Gamma) = E^{\text{int}}(\Gamma) + E^{\text{ext}}(\Gamma) \tag{2.4}$$

where E^{int} is the internal energy, and E^{ext} is the external energy.

Internal energy is based on the geometric information of the curves

$$E^{\text{int}} = \alpha \int_{\Gamma} \Gamma' \mathrm{d}\Gamma + \beta \int_{\Gamma} \Gamma'' \mathrm{d}\Gamma \tag{2.5}$$

where the first term is the first derivative that limits curve stretching, and the second term is the second derivative that minimize curve bending. α and β are constant coefficients.

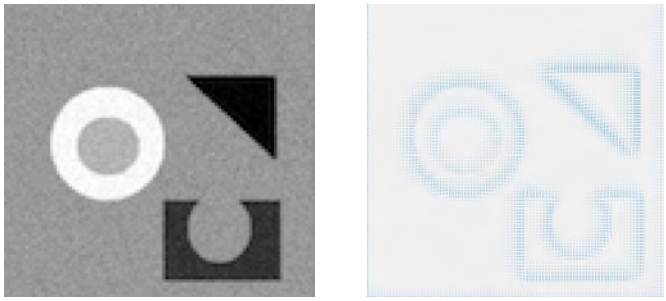
External energy is based on image data I

$$E^{\text{ext}} = \int_{\Gamma} P(I) \mathrm{d}\Gamma \tag{2.6}$$

where $P(I)$ is the potential energy function. There are many proposals for $P(I)$: Gaussian potential model [83, 157, 158], pressure model [38], distance potential model [39], gradient vector flow [171], and dynamic distance model [54, 99]. We shall briefly discuss the two most popular models, which are the Gaussian model and the gradient vector flow model.

Gaussian potential model is described as

$$P(I) = -\omega \left\| \nabla\big(G_{\sigma}(x,y) \circledast I(x,y)\big) \right\|^2 \tag{2.7}$$



(a) Sample image **(b)** Gaussian forces

Figure 2.5: Forces derived from Gaussian potential

where G_σ is a two dimensional Gaussian with standard deviation σ, $\circledast$ denotes the convolution operator, and ω is a constant coefficient.

Gradient vector flow diffuses the gradient of an edge map far from the boundary. One simple example to compute the velocities is

$$\begin{aligned} \mathbf{v}_0 &= \nabla I \\ \mathbf{v}_{k+1} &= \mathbf{v}_k + \mu \nabla^2 \mathbf{v}_k - \|\nabla I\|^2 (\mathbf{v}_k - \nabla I) \end{aligned} \tag{2.8}$$

where μ is a positive scalar, and k denotes iteration. The image I can be replaced with an edge map for better results.

Despite the variety, most external potential models use local differential properties of image edges, hence the segmentation may stick to local minimums. User-driven forces may be required to achieve desired segmentation [83]. Besides, they are not suitable for noisy images or weak edges. To overcome the issue, many authors utilize a global energy function proposed by Mumford and Shah [118], one of the most popular models in deformable model with many applications [8]. We shall discuss the Mumford-Shah energy function in the following section.

2.1.4 Mumford-Shah energy function

The Mumford-Shah functional [118] defines a criterion for approximating an image $I : \Omega \to \mathbb{R}$ with a piecewise smooth function $u : \Omega \to \mathbb{R}$ and a boundary set $\Gamma \subset \Omega$. The energy function to be minimized is

$$E(u, \Gamma) = \int_{\Omega - \Gamma} (\nabla u)^2 \mathrm{d}\Omega + \alpha \text{ length}(\Gamma) + \beta \int_\Omega (u - I)^2 \mathrm{d}\Omega \tag{2.9}$$

where α and β are the weights of the boundary length term and the smoothness term.

In a simpler version, the approximating function u is piecewise constant, leading to a partition of the domain into disjoint phases $\{\Omega_n\}$ of constant intensity $\{c_n\}$. The approximating function is now

$$u(\mathbf{x}) = c_n \text{ if } \mathbf{x} \in \Omega_n \,. \tag{2.10}$$

With a piecewise constant function, the boundary Γ is given by the partition, and ∇u vanishes inside each phase, yielding the reduced functional

$$E(u) = \sum_{n=1}^{N} \int_{\Omega_n} (c_n - I)^2 \mathrm{d}\Omega + \alpha \mathrm{Length}(\Gamma) \tag{2.11}$$

Existence of the solution for Eq. 2.11 is proved theoretically in [118, 107]. Also, finding the global minimum is a NP-complete problem [86].

Minimizing the Mumford-Shah functional using Eulerian mesh (or implicit representation) is popular due to its triviality in handling topology changes, and there are many proposals [13, 28, 29, 14, 4, 150, 86, 107, 31, 163, 140, 161]. Perhaps, the most popular model is the active contour without edge, a two-phase segmentation [31].

In contrast, there is few researches accommodate the Mumford-Shah energy function with explicit representation because of the difficulty in handling topological changes. Even if they do [58], they do not fully minimize the Mumford-Shah function due to the fact that they only have the information of the curves/surfaces without the interior/exterior regions, which will lead to holes and unconnected regions being not segmented unless the segmentation start with a good initialization.

2.2 Other image segmentation methods

Classically, image segmentation is defined as the partitioning of an image into nonoverlapping, constituent segments $\Omega = \bigcup_{i=1}^{N} \Omega_i$ that are homogeneous with respect to some characteristics such as intensity or texture [77, 75, 128]. Popular segmentation methods that are related to our approach are: thresholding, region growing, classifier, clustering, Markov random field (MRF), deformable model, and atlas-guided. Among them, thresholding, classifier, clustering and MRF

are pixel classification methods. Readers can refer to [131, 77, 77] for general surveys.

Thresholding is the basic approach in image segmentation. In simplest form, the method replaces each pixel with 0 or 1 if the intensity is larger or smaller than a fixed value called threshold (2.6b). Choosing thresholds is often based on histogram of intensity (2.6c). Though the process can be automatic [139], it is usually done with visual assessment of the result.

Thresholding is efficient for images with good contrast and low noise. The main problem of this approach is that only single pixel is considered without relationship with its neighbors, hence it makes the segmentation unconnected and sensitive to noise and inhomogeneities. For this reason, thresholding is often used to produce initialization for other methods.

(a) Lena 500 × 500 **(b)** Threshold at 102 **(c)** Histogram of intensity

Figure 2.6: Thresholding methods

Region growing iteratively evaluates neighboring pixels of segmented regions and decides to add the pixels based on some criteria such as intensity or edges in the image. This is a simple concept but can extract accurately the regions that have same properties. The limitation of the method is that it needs initial seed pixels, which requires manual input. Besides, it is a local method and is sensitive to noise.

Classifier is a supervised method that requires training data (pixels with known labels) as the references in order to label new data. The algorithm is based on feature similarity, and there are different models for classifiers: nearest-neighbor that uses intensity difference, k-nearest neighbors that generalizes the nearest-neighbor, maximum-likelihood that uses distribution of training data to compute probability of pixel belong to a label, etc. Because it does not require

iterative computation, classifiers can be efficient. However, the method can also be expensive as it has to store the whole training data. The weakness of this method is the training data, which may be time consuming to get and be affected by human error.

Clustering is basically classifier without training data. It is an unsupervised method that iteratively trains itself using current data. A popular algorithm in clustering is K-means [40], which, in each iteration, computes the mean intensity of each class and classify pixels using closest intensity. Though it is an unsupervised method, clustering still requires an initialization. Like classifier, clustering methods neglect the correlation between neighbor pixels, which leads to noise sensitivity.

Markov random field is a statistical model that considers correlation between nearby pixels. The model is efficient because in real images, regions are often homogeneous; neighboring pixels usually have similar properties. MRF is often used in clustering algorithms and has been proved its robustness to noise [182]. The limitations of MRF are parameters choices and expensive computation.

Atlas-guided segmentation is similar to registration problem [100], where a new image is warped on segmented images to be used as references. This approach is suitable for segmenting many data sets with small differences like medical scans from various patients. The disadvantage of the method is the process to construct the atlases.

Chapter 3

The Deformable Simplicial Complex (DSC) method

This chapter describes the Deformable Simplicial Complex (DSC), an explicit interface tracking method. We also briefly describe other interface tracking methods to demonstrate the properties of the DSC in comparison to its counter methods.

3.1 The DSC method

DSC [110] is a method for explicit interface tracking, which has been applied to fluid simulation [114] and topology optimization [36]. The advantages of the DSC are multi-phase support (Fig. 3.1) and meshes that can be used for finite element method (Fig. 3.2). This section describes the DSC data structure and algorithm.

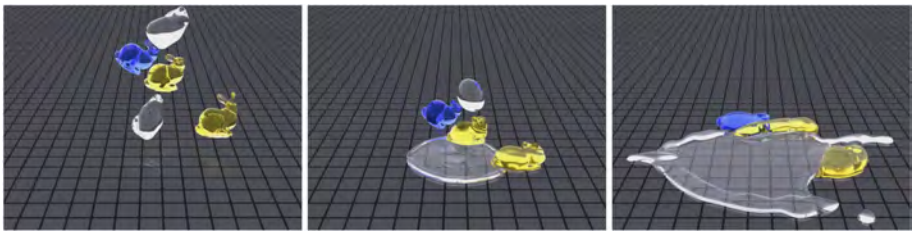
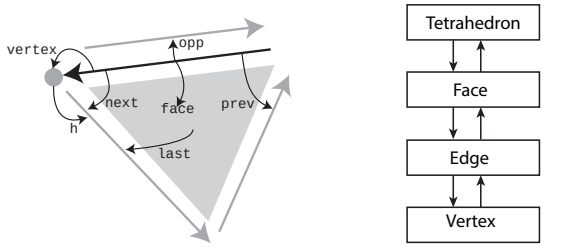


Figure 3.1: Applications of the DSC in fluid simulation. Image courtesy [114]

Figure 3.2: Applications of the DSC in topology optimization. Image courtesy [36]

3.1.1 The DSC data structure

The DSC is available for two and three dimensional problem. In 2D, it utilizes a triangle mesh with half-edge data structure (see 3.3a). In 3D, the DSC uses a tetrahedral mesh with index-based data structure (see 3.3b) that stores boundary and co-boundary of each element. Boundary of an element x consists of one-level-lower entities in the closure of x. Co-boundary of x is the dual of boundary, which includes one-level-higher entities whose boundaries contain x.



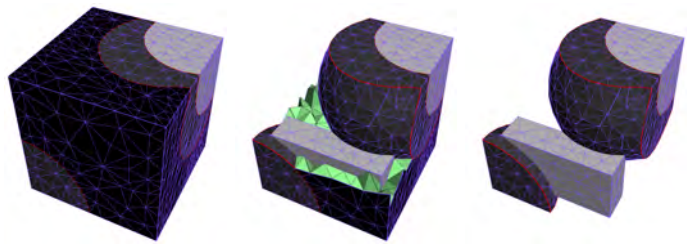
(a) Half edge in 2D **(b)** Index-based in 3D

Figure 3.3: DSC data structure

The DSC represents multiple phases by labeling volume elements (triangles in

2D and tetrahedra in 3D). Edges/triangles whose coboundaries have different label define the interfaces. Fig. 3.4 illustrates the DSC representing three phases in 3D.

The DSC utilizes topology relations to resolve intersections of interfaces, hence it does not know what happen outside the mesh domain. For this reason, the boundary of the mesh is fixed to avoid self intersections. In other words, the DSC is bounded by rectangular/cuboid domain.

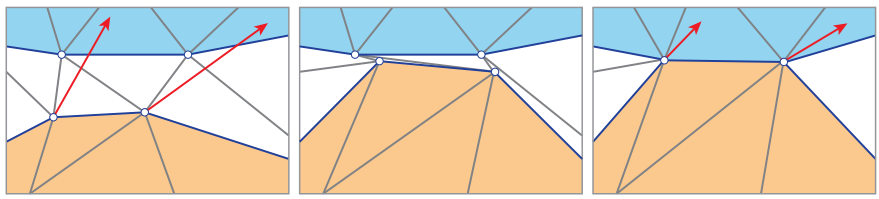


(a) Cuboid domain **(b)** A cross section **(c)** Phase 1 is hidden

Figure 3.4: 3D DSC representing three phases: Dark, gray and light. Green triangles are internal triangles, blue edges are interface edges, and red edges are junction edges

3.1.2 The DSC algorithm

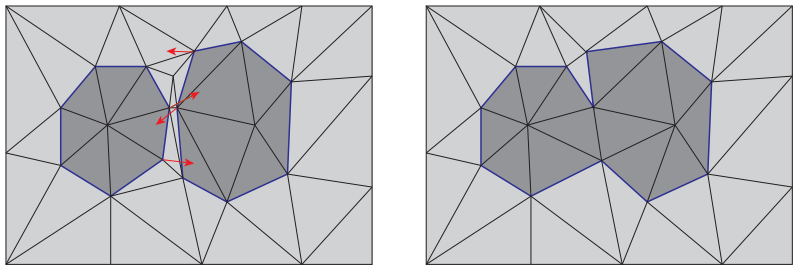
The DSC takes interface vertex displacements as input, deforms the mesh, and resolves topological changes automatically. An implementation of the DSC in C++ is available to publicity [7].



(a) Three-phase DSC **(b)** Vertices move inside their stars **(c)** After refinement

Figure 3.5: The DSC handles multi-phase colliding. Red arrows denote desired displacements

The DSC combines mesh refinement and topology fixer, so it can deform the interfaces while assuring the quality of the mesh using one algorithm. The DSC algorithm includes two steps. First, it moves the interface vertices as far as possible in their one-rings (neighbor triangles/tetrahedra) without making self intersection. After this step, the mesh is still valid but contains degenerated elements (see 3.5b). In the second step, the DSC applies mesh refinement to remove these degenerated elements and to maintain the quality of the mesh. Because the vertex displacements are limited inside their one-rings, the DSC may require some iterations to move the interfaces to the destination defined by user.



(a) Desired displacements **(b)** Moved mesh

Figure 3.6: The DSC handles merging

Algorithm 1: DSC algorithm

Input: mesh $\mathcal{M}$, displacements of interface vertices

1 **while** *not all vertices moved to their destinations* **do**

`/* Step 1: Move the interfaces */`

2 Move the vertices as far as possible inside their one-rings

`/* Step 2: Refine the mesh */`

3 Smooth

4 Mesh improvement

5 Remove degenerated tetrahedra (3D only)

6 Remove degenerated triangles

7 Remove degenerated edges

Alg. 1 shows the DSC algorithm. The mesh refinement process includes mesh smoothing, mesh improvement and degenerated element removal. The mesh improvement techniques are maximize the minimal angle in 2D and multi-face retriangulation in 3D [111]. The DSC uses volume-length ratio [129], min angle of triangle, and edge length to measure qualities of tetrahedra, triangles, and edges, respectively, and only locally refines the elements with low qualities. As the result, the DSC maintains the general topology of the mesh (See Fig. 3.6),

which is important for incorporating with finite element method and for reducing diffusion. Though it consists of different refinement techniques, the DSC algorithm utilizes only two basic topological operators, which are edge collapse and edge split, for all procedures.

3.2 Other explicit interface tracking methods

As we are concerned with both 2D and 3D problems, we shall focus on explicit interface tracking methods that support 3D tracking. Though the major concerns often be the performance and accuracy, we found that interface representation (triangle mesh or tetrahedral mesh) and multiple phases support are more important for image segmentation. We categorize explicit interface tracking methods to three groups: triangle-based with two phases, triangle-based with multiple phases, and tetrahedron-based.

3.2.1 Triangle-based with two phases

Majority of explicit interface tracking methods belong to this category due to the simplicity of a triangle mesh. A closed interfaces also implicitly define two phases as inside and outside, hence methods in this category do not require a technique to represent phases.

Boolean-like technique is probably the most intuitive approach to resolve mesh intersection. Boolean algorithm follows three steps: first compute the explicit intersection between meshes; second triangulate the intersecting triangle; and last remove all triangles that are inside. To determine if an element is inside, Campen and Kobbelt [23] use binary space partitioning (BSP) that decompose space into cells and label these cells, and Zaharescu *et al.* [181] use winding number (by casting a ray from a point and count the number of going inside or outside the mesh). These methods provide accurate interface, however resolving every single intersection is expensive.

Element deletion methods are proposed to reduce the complexity of resolving intersections. These methods follow two steps. First they detect the conflicting triangles (intersecting triangles and inside triangles) and delete these triangles. This step leaves holes on the meshes. The second step will fix these

holes by mesh generation. There are three approaches for element deletion: *proximity merging*, *hybrid with Eulerian grid*, and *intersection detection*.

Proximity merging methods [89, 155] utilize a quasi-uniform triangle mesh that keep the edge length uniform. They enforce small displacements to keep the mesh intersection-free. Vertices in proximity will be removed with their 1-rings, and the holes result from removal are joined by triangle strips.

Hybrid methods utilize Eulerian grids and update the singed distance field when the triangle meshes deform. The interfaces are then updated based on the isosurface of the singed distance field. Early methods [108, 109, 9] replace the whole interface by a new isosurface, which leads to heavy diffusion. Later proposals [56, 73, 74, 167] remesh the surface locally on affected grid cells hence avoid diffusion. However the accuracy still depends on the size of the grid cells. Some authors try to preserve the small or thin details by improving marching cube with marching cube convex hull [168] and marching template [116]. Though they can maintain small features, merging small features is less accurate, and splitting small features is not possible.

Intersection detection methods [15] detect intersecting triangles explicitly by triangle overlap test. Chentanez *et al.* [33] also detect inside triangles by ray casting test. After removing the conflicting triangles, they fill the holes with or join proximity open-curves by triangle strips. In their experiments, they claim a better performance than hybrid methods.

Comments: A common property of methods in this category is that they do not consider the motion of the mesh. This could lead to surface passing each other and limit the support for multiple phases as the algorithm could not compute the shared interfaces between colliding phases. Besides, they generally require uniform mesh with small edge length to guarantee the accuracy when they remove intersecting triangles.

3.2.2 Triangle-based with multiple phases

To our knowledge, there is only one approach [18, 45] in this category, which employs collision handler to push the intersecting surfaces to intersection-free state. Their next step, which handles topology changes, is similar to proximity

merging methods. To enhance the accuracy, the authors utilize continuous collision detection (CCD) that make use of the motion of the mesh. This method is highly accurate, however CCD becomes the bottleneck in their performance.

3.2.3 Tetrahedron-based methods

Tetrahedron-based methods naturally support multiple phases as the tetrahedra can be labeled to represent different phases. However, due to their high complexity, methods in this category are less exploited than triangle-based methods. Perhaps interface tracking using a tetrahedral mesh relates to the moving mesh problems, where objects, normally with fixed interfaces, move in a tetrahedral domain.

Quan *et al.* [135, 134] use label flipping to evade topology changes. By predicting the merging/splitting areas, the relabeling process will happen before the deformation of the mesh, hence it does not create interfaces conflict. Interior part of the mesh moves following optimization-based smoothing [49], which also creates no topology change. Their method is simple to implement but has limitations in the requirement of mechanisms to define merging/splitting areas, which makes it difficult to applied to interface tracking.

Pons and Boissonnat [133] use restricted Delaunay tetrahedralization to resolve the intersection. Once the mesh is moved, the restricted set is updated by removing elements in inverted tetrahedra, and a new Delaunay mesh is generated based on the updated interface. The labels of the volume elements in the new mesh are decided by projecting their circumcenters to the old mesh. This procedure requires the use of tree data structure to boost the search process. Because they do not utilize the motion of the interface, shared boundaries in case of multi-phase may not be precise, and, arguably, small objects may pass through each other. Besides, the lack of Steiner vertices may reduce the quality of the tetrahedral mesh hence make it be less useful.

3.3 The DSC in comparison to other tracking methods

The DSC belongs to the tetrahedron-based category. Tab. 3.1 show the comparison of the DSC with other methods. The most attractive properties of the DSC is multi-phase support and the ability to assess interior regions as it has a quality interior mesh.

Table 3.1: Explicit interface tracking methods comparison. The question mark '?' denotes possibility with further effort. Cat.1: two-phase triangle mesh; cat.2: multi-phase triangle mesh; cat. 3: tetrahedral mesh

Properties	Cat. 1	Cat. 2	Cat. 3	The DSC
Mesh data	triangle	triangle	tetrahedral	tetrahedral
Consider motion	-	✓	-	✓
Multi-phase	-	✓	✓	✓
No surfaces pass through	-	✓	-	✓
Unbounded domain	-	✓	-	-
Insertion of new regions	?	?	?	✓
Accuracy depends on	edge length	none	edge length	none
Adaptive resolution	-	✓	-	✓
Mesh for FEM	-	-	-	✓
Large displacements	-	✓	✓	✓
Simplicity	✓	-	-	-

Part II

Methodologies & Applications

Chapter 4

Cache-mesh, a Dynamics Data Structure for Performance Optimization

Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen
Technical University of Denmark

Published in *Procedia Engineering* 203, 2017.

Abstract This paper proposes the cache-mesh, a dynamic mesh data structure in 3D that allows modifications of stored topological relations effortlessly. The cache-mesh can adapt to arbitrary problems and provide fast retrieval to the most-referred-to topological relations. This adaptation requires trivial extra effort in implementation with the cache-mesh, whereas it may require tremendous effort using traditional meshes. The cache-mesh also gives a further boost to the performance with parallel mesh processing by caching the partition of the mesh into independent sets. This is an additional advantage of the cache-mesh, and the extra work for caching is also trivial. Though it appears that it takes effort for initial implementation, building the cache-mesh is comparable to a traditional mesh in terms of implementation.

4.1 Introduction

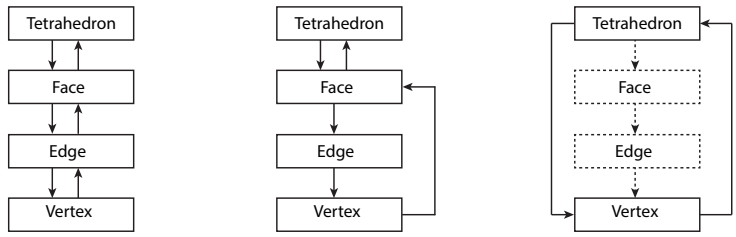
3D meshes are an essential part of computational geometry processing, with applications in finite element methods, deformable bodies, fluid simulation, topology optimization, visualization, etc. For mesh processing, we are generally concerned about mesh quality, memory usage and performance. All of these factors are important, but for dynamic meshes where geometry and topology change frequently, performance is often a very high priority. This paper discusses performance optimizations for 3D meshes which do not compromise the quality and add just a small increase in memory usage.

A mesh consists of entities, e.g. vertices, edges, faces and tetrahedra in a tetrahedral mesh. It also consists of a topology that describes the relations between these elements. There are many different types of topological relations, although, typically, only a few such relations are stored to ensure memory efficiency and simplify the implementation. The other relations must be calculated, and the complexity of this operation depends on the number of indirections from one mesh entity to the set of entities that we need. It can be time-consuming, especially in high-dimension meshes. To be more specific, we define $R(k)$ as the mapping from an entity k to the set of entities R that we seek. An analysis of the performance of $R(k)$ in ten common mesh data structures can be found in [71]. Fig. 4.1 shows three examples from [71], namely F1, F3 and R1, and Tab. 4.1 shows the memory operation counts (storage, retrieval, assignment and comparison of topological relations) of topology retrieval for these three meshes. We can see that the memory operation counts are significantly different, even though we only consider a subset of relations. In general cases, topology relations are more complex and memory operations vary even more.

Table 4.1: Comparison of topology retrival in memory operation count for three mesh data structures in Fig. 4.1, adopted from [71]. The notations V, E, F, T denote vertex, edge, face and tetrahedron, respectively.

Type	F(T)	E(T)	V(T)	T(F)	E(F)	V(F)	T(E)	F(E)	V(E)	T(V)	F(V)	E(V)
F1	4	36	30	2	3	13	50	5	2	619	399	14
F3	4	36	30	2	3	13	297	252	2	360	35	840
R1	72	58	4	302	24	3	214	721	2	23	3.4k	1.9k

We discuss the topology retrieval because it is one of the main factors that affect the performance. As mesh processing mainly deals with the topology, including inquiries, evaluation and modification, it leads to a large number of references to topological relations. Consequently, the data structure should provide fast retrieval of the most commonly used relations for a given problem.



(a) F1 mesh representation **(b)** F3 mesh representation **(c)** R1 mesh representation

Figure 4.1: Three data structures analysed in [71]. The arrow denotes stored adjacent relations. (a) Full one-level upward and downward adjacencies. (b) Full downward adjacencies and upward adjacencies from vertices to faces and faces to tetrahedra. (c) Reduced representation: Only vertices, tetrahedron and their adjacent relations in both ways are stored.

Unfortunately, this is not a straightforward process because the statistics of the topology retrievals differ greatly from one problem to another.

Generally, one avoids modifying the data structure of a mesh, because adding or removing one type of topological relation requires the modifications in all topological functions (functions that include topology changes). For this reason, the common approach for performance optimization is to select a suitable data structure, and this includes two steps: 1) profile the data references in the problem; and 2) then select a suitable data structure based on that statistic. There are limitations in this approach. First, because we are looking for something very specific, it may not exist, and often creating a bespoke data structure is not an option. Second, in order to profile the data reference, we need the problem to be implemented in advance. This means we solve and profile the problem with a pilot data structure, and then exchange it with an optimized data structure in the final phase. However, the extra effort for this replacement is also not negligible.

This paper proposes the cache-mesh; a dynamic mesh data structure that can be modified with trivial effort. The cache-mesh consists of a core mesh and a cache layer that stores extra topological relations. Our compelling advantage is the ability to change the types of stored topological relations at minor additional cost, which makes data structure optimization straightforward. Furthermore, the use of caching does not add much complexity if we decide to cache one more entity type.

Finally, yet importantly, we can store uncommon data, which rarely appears in

a mesh data structure. We will demonstrate an example in which caching entity attributes helps in enabling parallel mesh processing for 3D meshes. Though the cache-mesh may sound complicated to implement, it can be achieved with little effort using an existing mesh framework as the core mesh.

4.2 Related work

Meshes differ mainly in the types of topological relations they store. Based on this difference, the authors in [63] categorize general mesh data structures into three groups: incidence-based, which stores incident relations, including boundary and co-boundary entities; adjacency-based, which stores adjacent relations, including close, adjoining or neighboring entities; and edge-based, which considers edges primary and store their relations with other entities. However, possible data structures are much more varied, with many proposals from previous research. The reader can refer to [63] for a data structure for simplicial complex, [71] for a data structure of finite element analysis (FEA) applications, and [2] for common index-based representation. In this section, we will discuss some typical data structures that store different amounts of topological relations.

The most fundamental mesh structure consists of vertices, and the highest order elements with their vertices, as in 4.1c. For example, in a tetrahedral mesh we store the vertices, the tetrahedra and a topology that defines four vertices for each tetrahedron [146]. This data structure is compact, simple and suitable for finite element analysis which only queries the tetrahedra. Clearly, it is not optimal for problems that need topological relations, since these need to be inferred.

It is true that the higher the amount of stored topological relations, the better the performance [71]. However, this also raises the complexity in implementation as well as the memory usage. For this reason, mesh structures commonly stop short of storing all intermediate relations. Another example is the simplicial complex mesh [53], an incidence-based mesh that stores all boundaries and co-boundaries of the entities, as in 4.1a. Such simplicial complex mesh can be considered the top-performance data structure in practice. Between this representation and the fundamental representation, there are several proposals that store different entities and topological relations: Primarily downward adjacencies [82], reduced incidence-based [61], only downward adjacency [43], etc.

Another approach is edge-centered representation that stores the edges and their relations to other entities. As vertices and faces are directly related to edges,

this approach allows constant time adjacencies retrieval in 2D meshes. Several edge-centered data structures have been proposed: Winged-edge [10], half-edge [22], quad-edge [76], etc. For higher dimension, edge-based meshes are known for the advantages of oriented navigation, flexible modification, and the ability to generalize meshes in any dimension (e.g. Linear Cell Complex in CGAL [51, 52]). Unfortunately, the performance becomes a huge drawback because there is no direct connection from the low-order entities to the high-order entities. Some researchers try to overcome this problem by combining edge-centered, face-centered structures with additional incidence relations [88, 2, 27]. Again, this raises the question of which relation sets should be stored.

In [11], the authors provide a memory comparison of different data structures. The comparison covers a wide range of different methods, although there is a lack of consideration for the problems that utilize the mesh. For this reason, they do not have criteria for measuring the overall performance and are not able to provide a performance comparison. In [71], the authors try to put the comparison into a real use case. They estimate the computation time using the statistic of topology references from one problem in the MEGA software [151]. One certain limitation is that their conclusion may not hold true for other problems.

One idea to make the data-structure optimization straightforward is a dynamic structure that can change its data effortlessly. This has appeared in the literature. In [90], the authors propose a multiple adjacency data set that allows users to choose between four different levels. The higher level uses more memory but improves the performance. The limitation of this approach is inflexibility, as users cannot choose the other data set they would like.

CPU cache and geometry processing Caching is a technique that stores data for faster re-retrieval. The early idea of caching appeared in CPU architecture [55], when the gap between virtual memory access and register access increased. At software level, the CPU cache is utilized efficiently with compiler optimization [41] and cache-efficient algorithms [164].

Much research has considered CPU cache optimization for geometry processing with fixed meshes. The popular approach is to sort the data for optimal memory access. In [176, 177, 175], the authors propose a data layout for rendering and a boundary volume hierarchy for collision detection. In [80, 35, 94], CPU caches are mentioned for optimal rendering. In [141], the authors propose edge traverse for cache optimization, and also for rendering. Generally, a space-filling curve [138, 144] is utilized to store a cache-friendly layout. The limitation is that this can only be used for a fixed mesh. For a mesh that is dynamic, it is difficult to

utilize such a low-level CPU cache.

For a dynamic mesh, caching appears at a higher level and serves as precomputed data. Examples are the precomputed polygon surface of NURB in [91] for collision detection; and the precomputed distance field in [160], also for collision detection. However, they do not include the ability to change the topology of the mesh. Not as much research contains cache for mesh with topological changes, as this raises the complexity in implementation. In summary, a CPU cache-efficient geometry-processing algorithm is only available for fixed meshes. For dynamic meshes, cache appears as precomputed data and is limited to its specific problem.

4.3 Some terminologies

This paper is concerned with 3D meshes, and our experiments utilize a tetrahedral mesh. As discussed above, a mesh $\mathcal{M}$ contains entities linked by a topology. In geometry processing, we often refer to star and link in topology. For a single entity x, the star of x comprises all the entities that contain x. The closed star of x is the smallest subcomplex that contains the star of x. The link of x is the subtraction of the closed star and the star of x (Fig. 4.2).

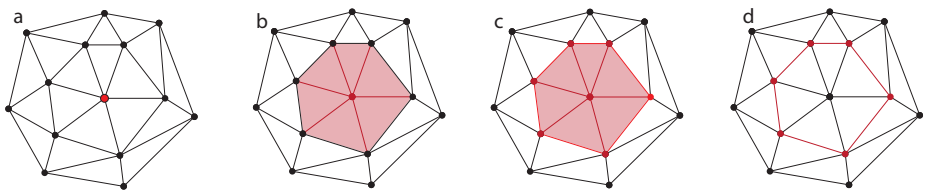


Figure 4.2: A vertex (a), its star (b), its closed star (c), and its link (d)

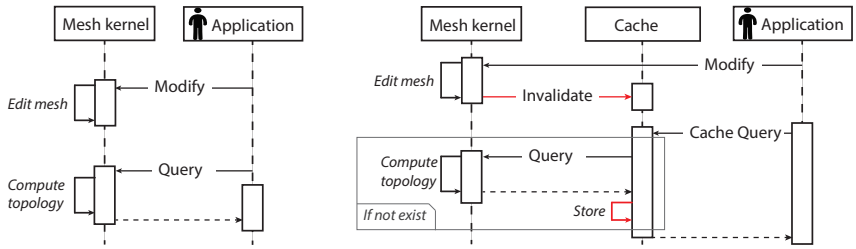
Commonly used topological relations are incidence and adjacency relations. Incidences are boundary and co-boundary: boundary of an entity x is the set of one-level-lower entities that belong to x; co-boundary of x is the set of one-level-higher entities whose boundaries contain x. Adjacencies are more general relations to entities which are close, adjoining or neighboring. For simplicity, meshes often store only first-order adjacencies (entities within the star of an entity, Tab. 4.2). In this paper, these commonly stored topological relations are called regular. In contrast, irregular relations are more complex and rarely appear in general mesh data structure.

Table 4.2: Complete list of first-order adjacencies (Ordered edges and ordered faces mean the entities are stored by a specific orientation)

Entity	First-order adjacencices
Vertex (V)	faces, edges, tetrahedra
Edge (E)	vertices, faces, tetrahedra
Face (F)	vertices, edges, ordered edges, tetrahedra
Tetrahedron (T)	vertices, edges, faces, ordered faces

4.4 The cache-mesh

In the following, we focus on the application of caching to 3D tetrahedral meshes. While e.g. triangle meshes might also benefit from caching, the utility is larger for 3D simplicial complexes since the number of possible relations is larger making it more expensive to maintain the full set of possible relations.



(a) Normal mesh **(b)** Cache-mesh

Figure 4.3: Components of a normal mesh and the cache-mesh

A mesh basically provides two functionalities: querying and modifying. Normally, query of un-stored topological relations leads to the computations of that data in every query, see Fig. 4.3(a). The cache-mesh has the same purposes but provides a cache query, where the data is kept for later access. The caching data can be any topology-related data, i.e. it may change when a topological event occurs. The cache-mesh is, in fact, a normal mesh with a cache layer that consists of the caching data and two functions: invalidation and storage, see Fig. 4.3(b). Sec. 4.4.1 will describe how the data is stored and how the storage function works; Sec. 4.4.2 will describe the invalidation process.

4.4.1 The cache component

A mesh data structure often stores mesh entities in arrays for optimal memory and performance. Any mesh entity is accessed by its array index, and this index is used as the unique ID for the entity. The mesh kernel must be able to provide topology references of any type, and we call this topology-retrieval function: $get\_data\langle data\_type\rangle(ID)$. Here, $\langle data\_type\rangle$ denotes a template which can change to any type of topological relations, and ID is the index of an entity. The $\langle data\_type\rangle$ can also be a topology-related attribute that changes when the topology is modified.

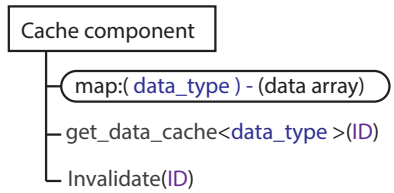


Figure 4.4: Data structure of the cache component

The cache component is an independent component, and it replaces some *get_data* functions. The structure of the cache component is shown in Fig. 4.4. We utilize a map to manage the caching relations, and the map is empty in the beginning. If a new type of topological relation is requested, we allocate a corresponding data_type and data array in the map.

Algorithm 2: *get_data_cache* function

Input: Mesh $\mathcal{M}$, cache $\mathcal{C}$, $data\_type$, ID

1 **if** $\mathcal{C}$.map[$data\_type$] *does not exist* **then**
2 | Allocate $\mathcal{C}$.map[$data\_type$]
3 **if** $\mathcal{C}$.map[$data\_type$][ID] *does not exist* **then**
4 | $\mathcal{C}$.map[$data\_type$][ID] = $\mathcal{M}$.get_data$\langle data_type\rangle(ID)$

Output: $\mathcal{C}$.map[$data\_type$][ID]

The cache component has two functions: *get_data_cache()* that replaces the normal *get_data()* function; and *invalidate()* to clean the outdated cache. As mentioned above, the *get_data_cache* function is an upgrade from the normal *get_data* function as shown in Alg. 2. The reference locality (miss/hit rate of the second if-then instruction) determines the performance gain, and it will be analyzed in Sec. 4.5.3.

The effort to implement this function is trivial as only little code is added to give the priority to the cached relations and to call the *get_ data* function if the data has not been cached. In fact, *get_ data_ cache* functions differ only in the data type of the caching relations. By utilizing template, we only need one template function, and other *get_data* functions can be upgraded to *get_data_cache* with one line of code. A sample implementation of the cache component in C++ can be found in Github [120].

4.4.2 Invalidate cache in common meshing procedures

Cache validation aims to maintain the accuracy of the caching topological relations. When a topology event occurs and changes the mesh, the cached relations may be different from the true data, hence the invalid cached data must be removed. The cache invalidation process is to find the affected entities in all topological functions. Though it may sound as if cache invalidation involves a lot of work, most geometry algorithms are the combinations of a few basic procedures, hence we only need to invalidate the cache in these basic functions.

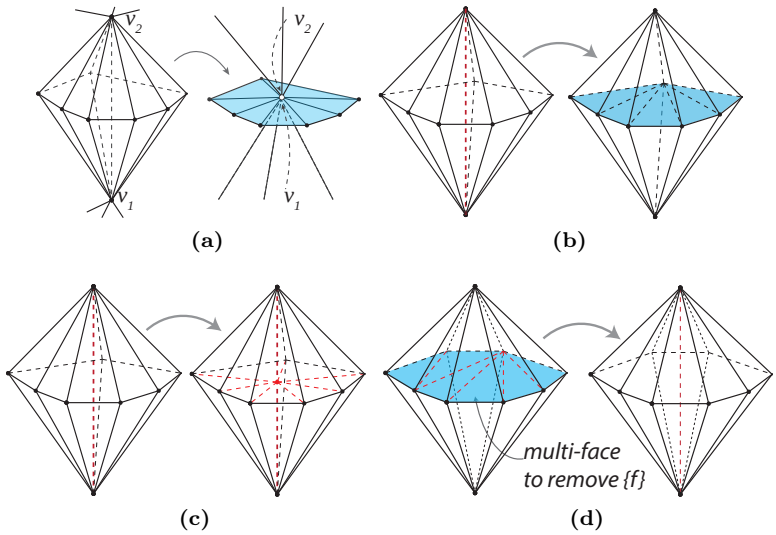


Figure 4.5: Common topological algorithms. (a) Collapse edge by merging vertices. (b) Collapse edge with minimal topological change. (c) Edge split. (d) Multi-face removal.

To find the affected entities, we find the affected tetrahedra in the meshing procedure. The affected entities are all entities inside these tetrahedra (including

vertices, edges, faces and the tetrahedra themselves). We limit the caching topology relations to the closed star of the key entity. Note that knowing the deleted/added entities is required in these procedures in order to update the stored relations in the kernel mesh, we can utilize this information to find the affected tetrahedra.

Common and basic 3D meshing procedures are edge collapse, edge split, and face flip [65, 85, 111]. The 3D face flip algorithm is the generalization of the 2D edge flip, and it is called the multi-face removal algorithm [152]. Demonstrations of these algorithms are shown in Fig. 4.5. The affected tetrahedra in the four algorithms in Fig. 4.5 are described in Tab. 4.3.

Table 4.3: Affected tetrahedra of common meshing algorithm in Fig. 4.5

Procedure	Affected tetrahedra
Merging vertices	Tetrahedra in the intersection of stars of v_1 and v_2
Collapse edge	Tetrahedra in the star of the collapsing edge
Edge split	Tetrahedra in the star of the splitting edge
Multi-face removal	Co-boundary tetrahedra of the faces that are being removed

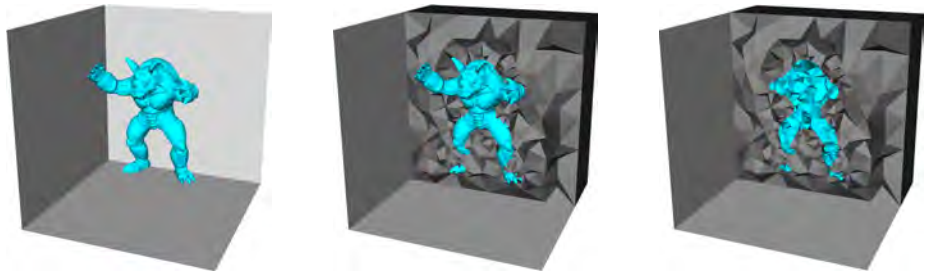
4.4.3 Utilizing the cache-mesh and profiling the references of topological relations

The utilization of the cache-mesh depends on whether the user-implemented functions contain topological changes in the mesh. The two situations will be described in two examples in Sec. 4.5.2 and Sec. 4.5.3. Generally, we follow three steps: integrate the cache component to the kernel mesh (omit this step if the cache-mesh is already integrated); profile the topology references; and update the *get_ data* functions of bottleneck topological relations to *get_ data_ cache*.

For profiling, our criteria are the number of calls and the computation time of the topology retrieval functions – one example is in Tab. 4.6. Of these two, the computation time has greater influence on deciding which topological relations are the bottleneck.

Another criterion for profiling is theoretical analysis of the algorithm, which is efficient for finding irregular topological relations references. These irregular relations should be within the closed star of the key entity since we limit the cache invalidation in the closed star.

4.5 Some practical examples



(a) (b) (c)

Figure 4.6: The Deformable Simplicial Complex (DSC) labels the tetrahedra to represent different objects. (a) The DSC interface, which represents an armadillo, in its domain. (b) The interface and a cross-section of the DSC domain. (c) A cross-section of the whole mesh.

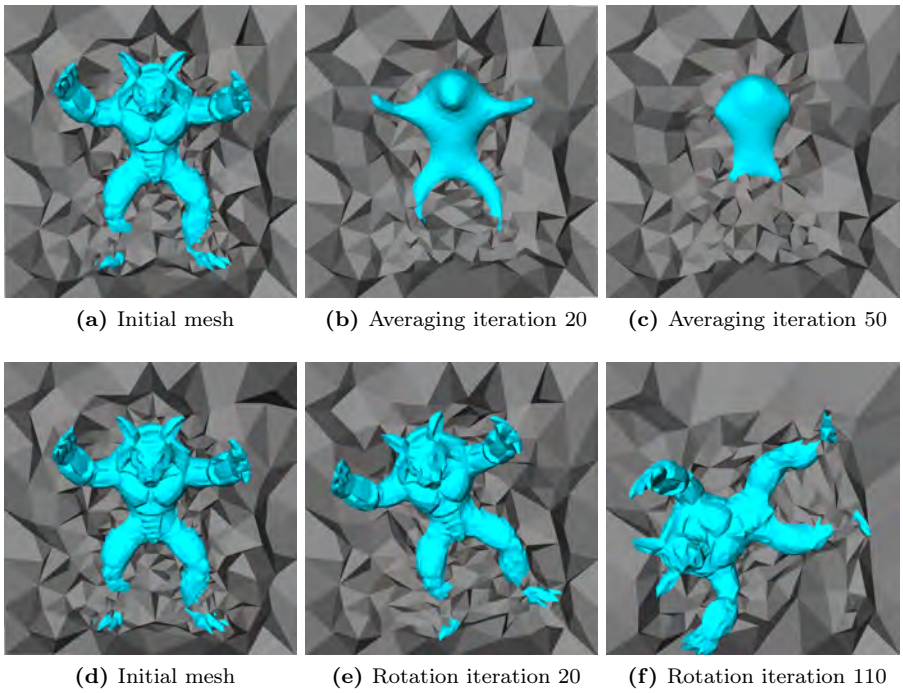
This section describes three examples of using the cache-mesh: quantitative information measurements (Sec. 4.5.2), where no topological event occurs; mesh processing (Sec. 4.5.3), where topological events occur; and parallel mesh processing (Sec. 4.5.4). This section will only shows methods and results, explanation of the results and discussion will be carried out in Sec. 4.6.

4.5.1 Experiment set-up

In all experiments, we utilize a tetrahedral mesh representing an armadillo. The specifications of the mesh are shown in Tab. 4.4. The kernel mesh is a simplicial complex mesh [53] that stores boundary and co-boundary of the entities as shown in 4.1a. When needed, the deformation and topology changes of the test object are handled by Deformable Simplicial Complex (DSC [110]), an explicit interface tracking method. See Fig. 4.6 for armadillo in DSC domain. In Sec. 4.5.3 we optimize performance of DSC, and there we describe the DSC algorithm in more detail. We use rotation and averaging motion for the experiments, see Fig. 4.7.

The specification of the mesh is shown in Tab. 4.4. The specification of the experimenting computer: Scientific Linux release 6.4 (Carbon) with 4 cores 2.5GHz CPU; 16GB of RAM; CPU cache: $32K$ L1d-cache, $32K$ L1i-cache, $256K$ L2-cache and $30M$ L3-cache. We use the gcc 7.1 compiler with `-Wall -O3` flags.

We utilize "Oracle studio 12.5 performance analyzer" for all measurements. We perform each experiment three times, and we always observe consistent results.



(a) Initial mesh **(b)** Averaging iteration 20 **(c)** Averaging iteration 50

(d) Initial mesh **(e)** Rotation iteration 20 **(f)** Rotation iteration 110

Figure 4.7: Averaging motion (top row) and rotation motion (bottom row). The figures show a cross-section of the tetrahedral mesh that represent an armadillo.

Table 4.4: Mesh specification of the armadillo model (number of elements).

vertices	interface vertices	edges	faces	interface faces	tetrahedra
4,872	2,164	33,738	57,518	4,324	28,651

4.5.2 Example 1: Extracting measurements with the cache-mesh

We first demonstrates optimizing the performance of operations which do not involve topological changes. In this example we measure quantitative information from the mesh: object volume; surface curvature; gradient of volume with respect to the displacement of the interface vertices; and energy change due to

vertices displacements (the energy requires the information of the region around the vertex). These 4 measures are collected in between the (not-optimized) DSC-handled deformations of the object under an averaging motion and a rotation motion, resulting in 8 experiments.

Table 4.5: Caching topological relations in the four measurements in the example 1

Quantitative information	Key entity (k)	Referent relations (R)
Compute volume	Tetrahedron	Vertices in the link
Compute surface curvature	Vertex	Interface vertices on the link
Compute volume gradient	Vertex	Interface edges on the link
Compute energy change	Vertex Vertex	Adjacent tetrahedra Faces on the link

To utilize the cache-mesh, we follow three steps in Alg. 3 (whereas there are four steps with traditional meshes in Alg. 4). In the Alg. 4, though the cache-mesh is not utilized, profiling is still necessary in order to select/build a data structure that fits the problem. In the current examples, the algorithms are known in advance, therefore we can analyze the set of the bottleneck topological relations theoretically. Tab. 4.5 shows the most-referred-to topological relations for each procedure. By storing these relations, we achieve up to 80% reduction in the computation time (Fig. 4.8).

Algorithm 3: Optimize the performance for a problem with the cache-mesh

1 Solve the problem with the cache-mesh
2 Profile the topology references
3 Update *get_ data* of bottleneck topological relations to *get_ data_ cache*
`/* trivial */`

Algorithm 4: Select / build a traditional mesh that optimizes performance for a problem

1 Solve the problem with a pilot mesh
2 Profile the topology references
3 Select / build an optimal mesh
4 Replace the pilot mesh with the optimal mesh

To compare the effort in optimization between the cache-mesh and traditional meshes, we shall discuss the Alg. 3 and Alg. 4. Utilizing the cache-mesh requires mainly two steps, and they are similar to the first two steps of optimizing a traditional mesh. On the other hand, optimizing a traditional mesh requires

two additional steps, and these steps are not trivial since implementing a mesh data structure includes a lot of work.

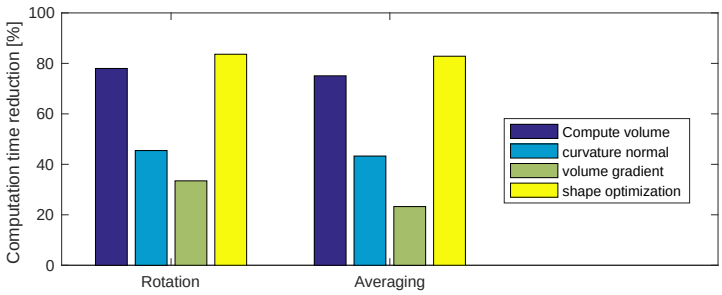


Figure 4.8: Performance gain in quantitative information measurements by caching topological relations

As the statistic of the topology references is known in advance, one may question the necessity of the cache-mesh. In fact, the quantitative measurements vary, and it is not possible to store all topological relations that they need. Furthermore, problems commonly do not utilize a single procedure but combine several procedures, and, in our experience, this fact leads to the differences in analytical profiling and practical profiling. To optimize the performance with a traditional mesh, we commonly need a pilot mesh (Alg. 4). On the other hand, the cache-mesh could adapt the cache to store different topological relation, and the effort for applying the cache is trivial. This case is useful for a public mesh framework that aims to serve many users with different requirements.

Algorithm 5: The DSC interface tracking

Input: Mesh $\mathcal{M}$, displacements of interface vertices

```
1 begin
2     while Not all vertices are moved to their destination do
3         Move interface vertices as far as posible
4         Mesh refinement
5             Smooth
6             Topological edge removal
7             Multi-face removal
8             Short edge collapse
9             Long edge split
```

4.5.3 Example 2: Tuning mesh processing with the cache-mesh

This example demonstrates optimizing performance of meshing procedures that involve topological changes. Here we optimize the performance of the interface tracking method, Deformable Simplicial Complex (DSC). The DSC tracks the deformable interfaces by iteratively moving the interface vertices without making inverted tetrahedra. Between displacement phases, the DSC applies a mesh refinement procedure to maximize the mesh quality. The DSC algorithm is described in Alg. 5. The implementation of the DSC in C++ can be found in Github [6].

Table 4.6: First-order adjacency references in the Deformable Simplical Complex. (a) Adjacent entities (key entities) (b) Queries count (millions) (c) Computation time (seconds) (d) Caching. The * denotes the stored topological relations in the kernel mesh.

(a)	F(T)*	E(T)	V(T)	T(F)*	E(F)*	V(F)	T(E)	F(E)*	V(E)*	T(V)	F(V)	E(V)*
(b)	37.8	0.25	16.7	62.5	80.4	75.7	0.35	24	165	0.95	0.81	5.6
(c)	14.6	0.8	78.6	20.7	28.5	136.6	0.1	8.4	57.2	39	6.7	2.1
(d)			✓			✓				✓		

Table 4.7: Irregular topology relations in DSC for caching. V, E, F denote vertex, edge and face.

Key entity (k)	V	V	E
Referent relations (R)	F in link	V in link	Sorted opposite E

Because the DSC contains topological changes, the process to apply the cache-mesh is a bit different from Alg. 3 in the first step. To apply the cache-mesh for the DSC, we follow three steps:

1. *Update topological functions to invalidate the cache:* There are five functions in Alg. 5, fortunately they are the combinations of only three basic functions: edge split, edge collapse and multi-face removal. Finding affected tetrahedra in these three functions is described in Sec. 4.4.2.

2. *Profile the topology references:* We measure reference count and computation time (as described in Sec. 4.4.3) of first order adjacency retrieval functions (Tab. 4.6). We also analyze the DSC algorithm theoretically to find the irregular topology references (Tab. 4.7).

3. *Apply cache:* We upgrade the retrieval functions of the relations (*get_ data*)

from the above step to the cached version (*get_ data_ cache* in Alg. 2). The effort for this step is trivial.

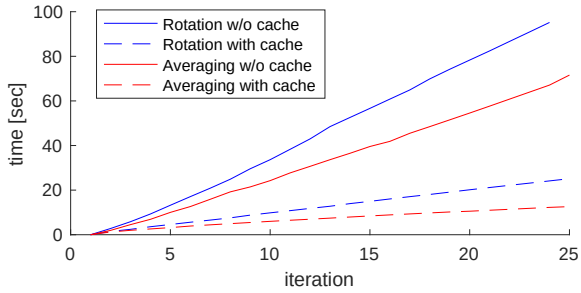
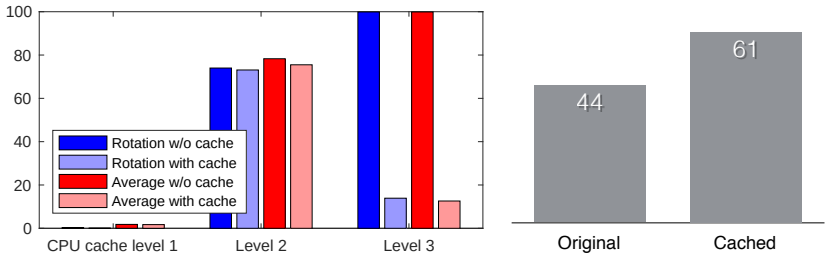
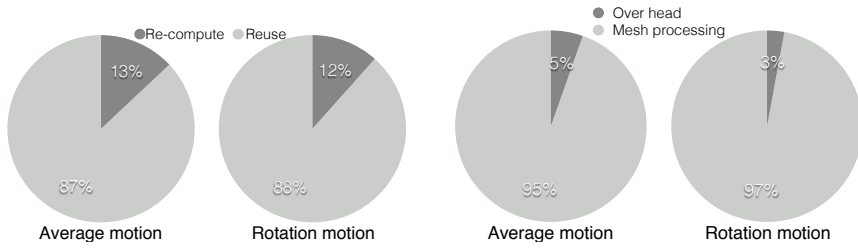


Figure 4.9: Computation time comparison



(a) Cache miss in system cache (CPU cache) (%) **(b)** Memory (MB) comparison

Figure 4.10: The CPU cache statistic and memory utilization



(a) Cache-mesh re-computation statistic (miss/hit rate of the second if-then instruciton in Alg. 2)

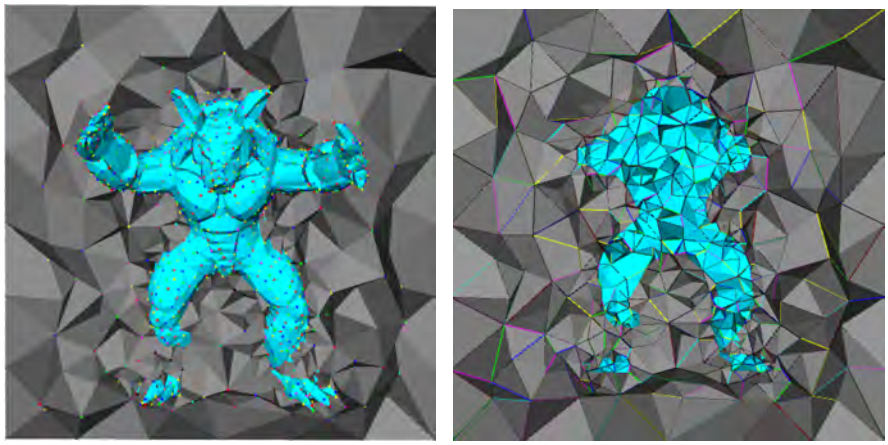
(b) Caching overhead of the computation time for invalidating the cache

Figure 4.11: Cache hit and cache over head

The effort for invalidating cache is probably similar to adding one type of topo-

logical relations to the data structure, but we then add six types of topological relations with trivial extra effort. Fig. 4.9 shows the results with performance improvement and other factors including: the CPU cache miss in Fig. 4.10(a); memory usage overhead in Fig. 4.10(b); reuse statistic of the caching topological relations in Fig. 4.11(a); and the overhead of the computation time for the invalidating cache. The overhead includes the time for finding affected elements and the time for memory allocating, storing and retrieving the cache data; Fig. 4.11(b). Generally, we observe five times in performance gain with 50% memory overhead. We discuss these results in detail in Sec. 4.6.

4.5.4 Example 3: Parallel mesh processing with the cache-mesh



(a) Vertices colors for parallel smoothing **(b)** Edges colors for parallel edge removal

Figure 4.12: Parallel mesh processing by coloring method

Parallel algorithms can be categorized into two groups: Distributed memory, in which a mesh is partitioned into parts that are independent in both memory and processing [147]; and shared memory, in which the partition only needs to be independent in processing. This section discusses a shared memory parallelism for mesh processing, and we employ the popular coloring method [67]. This method assigns different colors to independent entities (E.g. the independent colors of the vertices and the independent colors of the edges in Fig. 4.12(a) and 4.12(b)) so that entities with the same color can be processed in parallel. Though finding optimal colors is an NP-hard problem [70], independent sets can be defined by an efficient heuristic function [81, 97], and it works well for 2D meshes. For general 3D dynamic meshes, it is still not possible to color the

mesh iteratively as the time overhead for coloring is higher than the time for normal serial processing.

Algorithm 6: Coloring method

```
1 Get colors of all processing entities
2 S = {s_i}, s_i is the set of entities with same color
3 for each s_i ∈ S do
4 |   Start a thread to process s_i
5 Wait for all threads to stop
```

Algorithm 7: Get color of entity

Input: Mesh $\mathcal{M}$, entity x

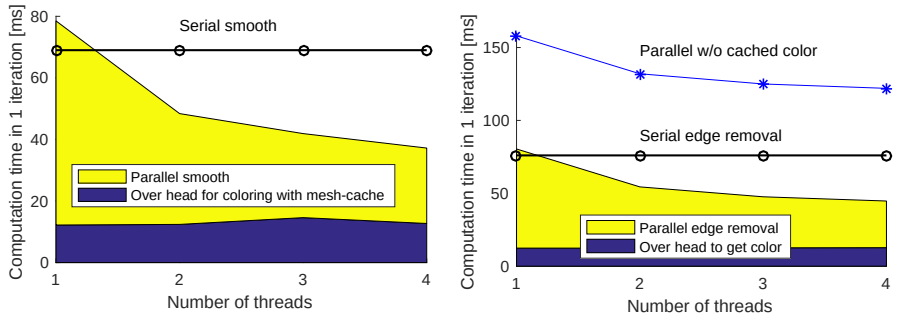
```
/* The related entities depend on the actual procedures        */
1 array S = Get color of related entities of x
2 if S is empty then
3 |   color ⟵ 1
4 else
5 |   color ⟵ min number which is not contained in S
```

Output: color

The cache-mesh enables coloring method for 3D dynamic meshes by caching the colors of the entities, and it reduces the overhead of coloring significantly. Define the color as an integer number, the algorithm of coloring method is shown in Alg. 6, and the algorithm for getting color of an entity is shown in Alg. 7. We apply caching for the colors for two meshing procedures: mesh smoothing [66] (the related entities are the neighbor vertices); and edge removal [152] (the related entities are the same as the affected edges for the cache invalidation in 4.5b).

The performance gain is shown in Fig. 4.13. Because of the limitation of the testing computer, we perform the experiment with up to four cores, and the computation time reduces up to 50%. About the scalability: the performance scales almost linear from one to two threads. However, the speed up reduces with more threads, and one reason is that full four cores could not be utilized by a single process. For a concrete number of parallel efficiency, we plan to experiment the cache-mesh with large data and better hardware in our future work.

Note that in order to make it possible for parallel processing, the mesh kernel must allow simultaneous modifications. For index-based mesh, the container is often an array, and it may not be able to add or remove elements at the same time. We slightly modify the array so that the container includes a reservation



(a) Parallel smooth **(b)** Parallel edge removal

Figure 4.13: Computation time of parallel edge removal and parallel smooth [ms]. In figure (a), we do not plot the computation time of parallel smooth w/o cached color as it is significantly larger ($\approx 451ms$) than that of serial smooth ($\approx 70ms$).

buffer, as described in Fig. 4.14. When we remove elements, the memory is not deleted but changes the state to *buffer*. When we add elements, the memory will be allocated in the reservation area. This modification is necessary for parallel mesh processing even with or without the cache-mesh, and, in fact, it is not rare in implementation of dynamic meshes.

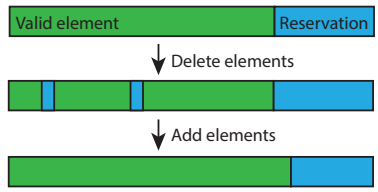


Figure 4.14: Kernel container for parallel processing

4.6 Discussion and conclusion

Our experiments show that if we store the topological relations, which are the most time-consuming to compute, we can reduce the computation time by up to 80%. This fact affirms that topology retrieving consumes the major computation resource in our testing cases. However, the references of topological relations differ greatly from one problem to another. Furthermore, they are not always regular for storing in a data structure. These two reasons make it difficult to optimize performance by fitting the data structure to the problem.

The cache-mesh provides a dynamic data structure that can adapt to arbitrary problems effortlessly. The advantages of the cache-mesh include:

- *Straightforward performance optimization:* Building an optimal mesh includes three steps with the cache-mesh, and the last step is trivial; whereas it takes four steps with traditional meshes, and none of them is trivial (Sec. 4.5.2).
- *Storing additional topological relations without increasing the complexity in implementation*: Since adding data is effortless with the cache-mesh, we can store additional topological relations for a better performance.
- *Storing irregular relations:* Some topological relations, e.g. opposite edges or some user-defined relations, are not stored in traditional meshes, otherwise the mesh data structures would be highly specific.
- *Parallel mesh processing:* Sec. 4.5.4 demonstrates the example where storing entity colors helps enable parallel mesh processing with minor extra effort.

The other facts of the cache-mesh, which can be considered advantages, are memory overhead and effort to implement from scratch that are comparable to a traditional mesh. The correlation of the cache-mesh and the CPU cache is also an interesting fact that we would like to discuss. In Fig. 4.10(a), the L3 cache seems to reflect the performance gain of the cache-mesh, which means it is not the instruction counter, but the latency in RAM has stronger influence on the computation time of the meshing procedures.

Regarding the performance gain, this depends on the portion of the affected entities during topological events. In our experiments of an deforming mesh, this portion is less than 15% (Fig. 4.11(a)), and caching reduces the computation time by around 80%. We also notice that the cache-mesh may not be as fast as traditional meshes that store the same topological relations because the caching data is not utilized inside the mesh kernel. Fortunately, this difference only manifests itself when the topological relations are recomputed (the 15% in our experiments). For dynamic meshes, the number of topological events should be controlled not only to reduce the number of entities affected by topology changes but also to increase the quality and accuracy of the mesh.

In conclusion, the cache-mesh provides a dynamic mesh data structure for convenient and effortless modification, which makes the performance optimization process straightforward. The cache-mesh boosts the performance in three ways: faster topological relations retrieval, as the relations are cached; possibility to

store more relations, as adding topological relations does not raise the complexity; and parallel processing, which is difficult for traditional meshes.

For further study, we plan to experiment the cache-mesh with more types of kernel mesh as well as providing a stand-alone cache-mesh that has an optimal kernel mesh. We also intended to analyze the parallel efficiency with large data and better hardware.

Chapter 5

2D Intensity-based Image Segmentation

Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen
Technical University of Denmark

In submission to *Pattern Recognition Letter*

Abstract This paper proposes a method for image segmentation using a deformable triangle mesh in the image domain. We define a piecewise constant function by labeling the mesh triangles with different phases, each representing a segment of an image. Our method finds the optimal mesh configuration and triangle labeling that minimize the piecewise constant Mumford-Shah functional. Contributions of this paper include a force model that moves mesh vertices towards the solution, and an adaptivity model that further adapts the mesh by introducing or removing vertices. The results demonstrate the advantages of our method over traditional methods like snakes and level set. Our approach supports multi-phase segmentation incurring no particular overhead. Furthermore, the use of an adaptive mesh facilitates accurate segmentation with a very compact representation. The biggest challenge of deformable meshes, changes to the topology of the segments, is handled by employing Deformable Simplical Complex (DSC), a method for explicit interface tracking.

5.1 Introduction

Image segmentation is one of the fundamental tasks in image processing. A wide range of methods have been developed for image segmentation, but here we focus on deformable models. The utility of this lies in the combination of information from the image with possible geometric constraints, providing regularization in the case of noisy images.

Typically, the curves defining the segmentation boundaries are represented in one of two ways: *explicitly* or *implicitly*. An explicit representation may be as simple as a sequence of connected line segments [156]. Perhaps, the most important advantage of explicit curves is that information about the segment geometry is given by the representation with no need for further processing. Unfortunately, changing the topology of segmented regions is generally very challenging. In implicit methods [125], one uses an auxiliary image to determine whether a given point belongs to a given segment. Changing the auxiliary image can be used to change both geometry and topology of the represented shapes. This representation is very effective if topological adaptivity is a concern, but it becomes less suitable if we need to segment an image into more than two labels.

In this paper, we advocate a novel approach to segmentation with an explicit representation of segment geometry. Like other explicit approaches, our representation makes it straightforward to compute geometric statistics such as boundary length, area, or curvature. But more than that, our method also allows the segment topology to change transparently, and there is no restriction on the number of labels that can be handled.

Our curve representation relies on a triangle mesh, where each triangle is given a label that indicates to which segment it belongs. The segmentation boundaries in this representation are the edges shared by triangles that have different labels. To deform a a segment, we only need to move the boundary vertices. In this sense, our representation is explicit except that topology changes can be handled by changing mesh connectivity and triangle labels. The underlying machinery for topological adaptivity is known as the DSC (*Deformable Simplical Complex*) method [110].

The use of a mesh representation also imbues our method with another important advantage, namely adaptivity. The mesh is allowed to be more dense in some regions than others, leading to a natural adaptivity unlike in a regular grid.

5.2 Related work and contributions

Deformable curves are popular tools for image segmentation. Following the original idea by [62, 166], various models have been proposed, see [172] for comprehensive overview. Different approaches vary primarily in the curve representation and in the dynamic model.

Regarding the dynamic model, early methods [83, 24, 106] use edge response computed from image gradient. Such forces are local and unsuitable for noisy images or weak edges. Region-based forces [31, 163] utilize a global information obtained from the segmentation regions to overcome these problems. Many global methods minimize the Mumford-Shah functional [118], a widely researched area [107, 183, 29, 58].

For curve representations, there are approaches using explicit curves [83, 38, 58] and those using implicit curves based on level sets [105, 26, 125]. Regardless of the representation the fundamental principles of deformable models are the same [173]. However, an implementation will inherit the limitations of the underlying curve representation. The choice of representation is therefore made by considering topological and geometric complexity of the problem.

Probably the most popular model with implicit representation is active contours without edges [31], originally formulated for two phases. For multi-phase segmentation, [183] use N level set functions for N phases, leading to simple equations but memory inefficient and suffering from phase overlap. [163] represent 2^N phase with N level set functions, which solves overlap problem but leads to more complex equations and may fail when multiple level sets evolve simultaneously. [96] use one level set function to represent N phases, but this approach leads to ill-conditioned equations as well as difficulty in extracting quantitative information like the unit normal and the curvature.

As for explicit curves, many methods attempt resolving topology change. One of the earliest examples is T-snakes [108, 109] which employs a fixed resolution grid to resolve topological changes during reparameterization. Unfortunately, this procedure diffuses interface details. In fact T-snakes is similar to implicit methods in this regard. Furthermore, this method provides no support for multiple phases. Similar methods for surfaces in 3D include [167, 181, 45].

Alternative approaches represent both the curves and the domains [132, 110]. This is convenient if the aim is to use global deformation forces, as in case of minimizing the Mumford-Shah functional. Still, only few methods combine an explicit curve and global forces, examples being [58] for two-phase 3D segmentation and [48] for multi-phase 2D segmentation. Those models use only

meshes of uniform resolution and do not fully exploit the advantages of explicit representation.

Adaptive mesh is utilized in a related, but rather different, segmentation approach [29, 28]. Instead of considering segmentation boundaries, they minimize the Mumford-Shah functional by assigning an intensity to each triangle. The mesh adaptively increases resolution at image features, resulting in smooth intensity transition at boundaries.

We extend the work of [48] and address its shortcomings. Just as [48], we use curve representation provided by [110] to handle topology changes. This gives us a number of advantages compared to implicit methods. First, we do not rely on the image grid, and can represent segmentation boundaries using any resolution, also changing adaptively across the image. Second, we can effortlessly handle an arbitrary number of segmentation phases and junctions between interfaces, which is cumbersome using level-sets. Multi-phase support is also our strongest advantage when compared to explicit methods. Furthermore, our topological adaptivity is straightforward, and not dependent on an auxiliary representation.

As a novel contribution, we derive a full dynamic model for minimizing the piecewise-constant Mumford-Shah functional on an adaptive, deformable triangle mesh. The new features of our model are:

- edge-based deformation forces, allowing us to correctly deform the mesh regardless of edge length, and leading to correct treatment of junction vertices;
- an adaptivity model, resulting in an accurate and compact capture of segmentation boundaries at all scales.

The results demonstrate the strengths of our approach, combined with a better performance than the level set method.

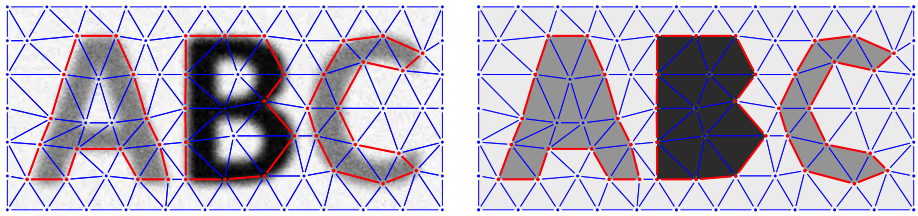
5.3 Method overview

Given the input image g, our primary objective is to find a piecewise constant u, represented on a mesh, that minimizes the Mumford-Shah energy in Eq. 2.11. Our secondary objective is to find a compact representation of u.

We represent u on a mesh $\mathcal{M}$ (defined by vertex positions $\mathbf{v}_i$ and the connectivity between vertices) with a triangle labeling L and phase intensities c_n. All those

will change during optimization, and to find the optimal u we will consider one entity at a time, while keeping the others fixed.

First, for a given mesh configuration and a labeling we find the optimal c_n (Sec. 5.4.1). Then we derive evolution forces for moving interface vertices $\mathbf{v}_i$ using gradient based method (Sec. 5.4.2). We employ DSC to deform the mesh, which also handles the topology change caused by the displacements.



(a) An image and a curve **(b)** A segmentation

Figure 5.1: Representation of interfaces and segmentation with the DSC in 2D. (a) Red edges represent the interface (b) The segmentation includes three phases: dark, gray, and white.

However, there are situations, illustrated in Fig. 5.1, which cannot be successfully resolved only by moving interface vertices and adjusting phase intensities. Therefore, we extend our algorithm by a number of energy-triggered discrete events. Part of this is an adaptive mesh subdivision, which locally adapts the resolution of the mesh.

One data-driven purpose of adaptive mesh is ensuring an accurate fit of the segmentation boundaries. For example, the mesh in Fig. 5.1 is too coarse to capture the curved boundary of the letter 'C'. To resolve such situations we define an interface edge adaptation (Sec. 5.5.2).

Evolving and adapting interface will not introduce new regions. This is handled by triangle relabeling, where a triangle moves from one phase to another (Sec. 5.4.3), as in the case of the hole in the letter 'A'. As for the letter 'B', triangles will be split (Sec. 5.5.1) prior to relabeling. Triangle split is data-driven mesh adaptation process which also ensures that we correctly capture small regions.

Finally, to improve the compactness of the representation, we collapse adjacent edges which are on a line and we merge inside triangles (Sec. 5.5.3). This mesh-driven process minimizes the mesh size without compromising the accuracy of the fit.

The discrete events are computationally expensive but occur rarely. Therefore

we adapt mesh once in every N_a iterations of mesh deformation. The optimal number of N_a will be discussed in the Sec. 5.7.

The general algorithm of our method is shown in Alg. 8. Initialization of the labels L is arbitrary. Throughout the paper we use random initialization but we sometimes show a manual initialization to demonstrate properties of our method. Segmentation runs until convergence, and the term *not converged* means that the residual error is still higher than a defined threshold.

Algorithm 8: Segmentation with deformable mesh

Input: image g, initial mesh $\mathcal{M}$ and init label L

```
while not converged do
    Compute {c_n}                                          // Sec. 5.4.1
    Compute interface vertex displacements // Alg. 9
    Deform the mesh M with DSC                               // Alg. 5
    if N_a iterations then                                 // Adapt mesh
        Adapt and relabel triangles                          // Alg. 10
        Adapt interface edges                                // Alg. 11
        Thin mesh                                          // Sec. 5.5.3
```

Our problem is a particular case of the reduced Mumford-Shah problem, whose minimizers exist [118]; therefore, we expect the existence of our solutions. Furthermore, we minimize the same energy function proposed by [31, 163], where minimizers exist due to the arguments of calculus of variations [3]. Our work is different as it utilizes a mesh to represent the curves, which has advantages in multi-phase problem.

Finding the global minimum is a NP-complete problem [86], and our solution is a local minimum that approximates the global minimum. The results may depend on the initialization of the labeling function. Our favorite initializations are multi-phase thresholding and random labeling, which in our experience of the provied desired segmentations.

5.4 Steps on a fixed-resolution triangle mesh

In this section we describe the energy minimization steps on a mesh of a fixed resolution: changing phase intensities, moving interface vertices and relabeling triangles. In the next section we extend the algorithm with the steps which locally change the resolution of the mesh.

5.4.1 Phase intensities c_n

Keeping the mesh $\mathcal{M}$ and the labels L of the triangles fixed, it is easy to find phase intensities $\{c_n\}$ which minimize (Eq. 2.11). Since c_n only influences phase Ω_n we have

$$\frac{\partial E}{\partial c_n} = \int_{\Omega_n} 2(c_n - g)\mathrm{d}\Omega\,. \tag{5.1}$$

This leads to phase intensity minimizing the energy

$$c_n = \frac{\int_{\Omega_n} g \ \mathrm{d}\Omega}{\int_{\Omega_n} \mathrm{d}\Omega}\,, \tag{5.2}$$

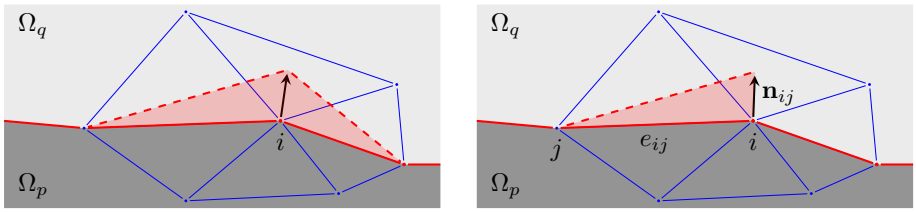
which is the mean intensity of the image g in Ω_n. The same result is derived in [31].

5.4.2 Interface vertex positions $\mathbf{v}_i$

Keeping the triangle labels s_t and the phase intensities c_n fixed, we minimize (Eq. 2.11) with respect to the vertex positions $\mathbf{v}_i$ using steepest descent method. Positions of interface vertices will be iteratively changed as

$$\mathbf{v}_i^{\text{new}} = \mathbf{v}_i + \frac{\partial E}{\partial \mathbf{v}_i}\mathrm{d}t = \mathbf{v}_i + \mathbf{F}_i\mathrm{d}t\,. \tag{5.3}$$

Here $\mathbf{F}_i$ is the force affecting vertex i and $\mathrm{d}t$ is the length of the time step. To compute the gradient of E with respect to vertex positions, we consider the two terms in (Eq. 2.11) separately.



(a) Vertex displacement **(b)** Contribution of e_{ij}

Figure 5.2: Computing energy change due to vertex displacement. **(a)** The red area is affected by the displacement of vertex i. **(b)** Contribution of e_{ij} is computed as a line integral over the edge.

External energy E^{ext} Tangential displacement does not change the external energy, so we only consider a normal displacement of a vertex. In particular, we consider a vertex i, and a directed interface edge e_{ij} between phases Ω_p and Ω_q, as illustrated in Fig. 5.2. We consider displacing vertex i for δ_i in the direction $\mathbf{n}_{ij}$, normal to the edge direction. We need to compute the change in external energy E^{ext}, c.f. Eq. 2.11, and we denote this change $\Delta E^{\text{ext}}_{ij}$.

In general, computing $\Delta E^{\text{ext}}_{ij}$ would require integrating over the triangular area A covered by the moving edge, since this is the area which contributes to the change in energy.

$$\Delta E^{\text{ext}}_{ij} = \int_A (c_p - g)^2 \mathrm{d}A - \int_A (c_q - g)^2 \mathrm{d}A \tag{5.4}$$

Assuming a small displacement, the area integral can be approximated by the line integral along e_{ij}, where ℓ is a distance from i. We have

$$\begin{aligned} \Delta E^{\text{ext}}_{ij} &= \int_0^{L_{ij}} \left[(c_p - g)^2 - (c_q - g)^2 \right] \frac{L_{ij} - \ell}{L_{ij}} \mathrm{d}\ell \, \delta_i \\ &= (c_q - c_p) \int_0^{L_{ij}} (2g - c_p - c_q) \frac{L_{ij} - \ell}{L_{ij}} \mathrm{d}\ell \, \delta_i \, . \end{aligned} \tag{5.5}$$

Here, L_{ij} is a length of the edge e_{ij}.

The change of energy being proportional with δ_i we can straightforwardly find the direction of negative gradient. We express this as a force affecting an infinitesimal segment of the curve, where we still consider e_{ij} on an interface between phases Ω_p and Ω_q,

$$\mathbf{f}^{\text{ext}}_{ij} = (c_p - c_q)\,(2g - c_p - c_q)\,\mathbf{n}_{ij} \, . \tag{5.6}$$

Note that this is an explicit curve counterpart of the implicit curve result from [31].

To compute the force affecting the vertex i, the force from (Eq. 5.6) needs to be interpolated and linearly weighted along the edge e_{ij}. Finally, an interface vertex i is adjacent to at least two interface edges. In case of a T-junction or crossing, three or more interface edges meet at a vertex. Therefore, the force affecting an interface vertex is a sum of contributions from all adjacent interface edges

$$\mathbf{F}^{\text{ext}}_i = \sum_{e_{ij} \text{ on interface}} \int_0^{L_{ij}} \mathbf{f}^{\text{ext}}_{ij} \frac{L_{ij} - \ell}{L_{ij}} \mathrm{d}\ell \, . \tag{5.7}$$

Estimating vertex forces by integrating along the interface edges as derived in (Eq. 5.7) is a significant improvement of the approach from [48], where forces are estimated only at vertex positions. This allows us to accurately capture the sharp features, and correctly handle T-junctions and crossings.

However, as in [48] we found that using forces (Eq. 5.6) leads to stability issues in multi-phase case. This is because the absolute value of the constant factor $(c_p - c_q)$ influence the overall speed of the curve evolution between phases Ω_p and Ω_q, but a different constant controls speed of the curve between another par of phases. Instead of introducing a different time step for every curve, we adopt the approach from [48] and use normalized forces

$$\mathbf{f}_{ij}^{\text{norm}} = \frac{1}{(c_p - c_q)^2}\,\mathbf{f}_{ij}^{\text{ext}} = \frac{2g - c_p - c_q}{c_p - c_q}\,\mathbf{n}_{ij}\,. \tag{5.8}$$

Internal energy E^{int} The length of the segmentation boundaries is sum of lengths of all interface edges, so we have

$$E^{\text{int}} = \alpha \sum_{e_{ij} \text{ on interface}} \|\mathbf{v}_j - \mathbf{v}_i\|. \tag{5.9}$$

The internal force derived from the gradient of E^{int} with respect to position of vertex i is

$$\mathbf{F}_i^{\text{int}} = \alpha \sum_{e_{ij} \text{ on interface}} \frac{\mathbf{v}_j - \mathbf{v}_i}{\|\mathbf{v}_j - \mathbf{v}_i\|} = \alpha \sum_{e_{ij} \text{ on interface}} \mathbf{f}_{ij}^{\text{int}} \tag{5.10}$$

Implementation In our implementation, line integral is computed by discretizing the edge to K segments, where segments are approximately 1 pixel long, as indicated in Alg. 9.

5.4.3 Triangle labels L

As mentioned in [48], evolving the curve by moving interface vertices will not introduce new regions, which is crucial when a phase contains holes or disjoint regions. To introduce new regions we relabel triangles. E.g. a triangle over the hole of the letter 'A' in Fig. 1.5 will be relabeled to belong to the background phase.

Simply minimizing Mumford-Shah energy by keeping the mesh $\mathcal{M}$ and phase intensities c_n fixed would result in labeling L based on mean triangle intensities.

Algorithm 9: Computing forces on interface vertices

Input: Image g, mesh $\mathcal{M}$, triangle labels s_t, phase intensities c_n

Allocate $\mathbf{F}_i = \mathbf{0}$ `// Array of vertex forces`

`/* External forces */`

forall the e_{ij} *on interface* **do**

  $K = \text{ceil}(\text{length}(e_{ij}))$ `// For discretization`

  $i, j =$ two vertices of e_{ij}

  $\mathbf{n}_{ij} =$ oriented normal of e_{ij}

  $c_p, c_q =$ intensities of two phases associated with e_{ij}

  for $k = 1 : K$ **do** `// Edge integral`

    Compute $\mathbf{f} = \frac{2g - c_p - c_q}{c_p - c_q} \frac{1}{K} \mathbf{n}_{ij}$ // (Eq. 5.8)

    $\mathbf{F}(i) = \mathbf{F}(i) + \mathbf{f}k$ // (Eq. 5.7)

    $\mathbf{F}(j) = \mathbf{F}(j) + \mathbf{f}(K - k)$ // (Eq. 5.7)

`/* Internal forces */`

forall the *vertex* i *on interface* **do**

  forall the e_{ij} *on interface* **do**

    $\mathbf{F}(i) = \mathbf{F}(i) + \mathbf{f}_{ij}^{\text{int}}$ // (Eq. 5.10)

This might lead to undesirable labeling when the triangle is inhomogeneous, i.e. it covers a part of the image with significant differences in intensity. An example is shown in Fig. 1.5 where triangles covering the black-and-white hole of the letter 'B' might be labeled as belonging to the gray phase. To avoid such situations, we only relabel the triangles that are homogeneous. In case of inhomogeneous triangles the situation will be resolved during mesh adaptation.

We measure the homogeneity of the triangle using intensity variance, which is also used for mesh adaptation. Therefore we combine the relabeling process and the mesh adaptation process. We will discuss this combination in Sec. 5.5.

5.5 Adaptive mesh

A big advantage of our explicit scheme is that we can adapt the triangle sizes as needed. In our approach, we start with a sparse mesh and then subdivide the mesh locally.

Local adaptivity results from the sequence of discrete events (splitting or merging) on mesh entities. Those events are triggered using a set of threshold values,

e.g. for length and energy. Adapting the mesh does not change the interface and is not directly contributing to minimizing the Mumford-Shah energy. Instead, adaptivity will assist further energy minimization in the subsequent steps.

An important threshold concerns stability. Adaptive mesh should only resolve the situations which cannot be resolved by moving interface vertices. Therefore we adapt only stable edges and triangles, *i.e.* not containing vertices evolving under energy-based forces. A stable vertex has displacement under a threshold value, typically set to 0.01 pixel.

Another important threshold is MIN_EDGE, a minimal edge length, which controls the finest mesh resolution.

5.5.1 Triangle adaptation

We propose using intensity variance on the triangle. Fig. 5.3 shows the histogram of the triangle energy in the segmentation of cement (Fig. 5.17). We can see that the energy function using the Mumford-Shah functional reports more triangles with high energy than the energy function using intensity variance does, and it is not correct. When the segmentation converges, the histograms become more similar.

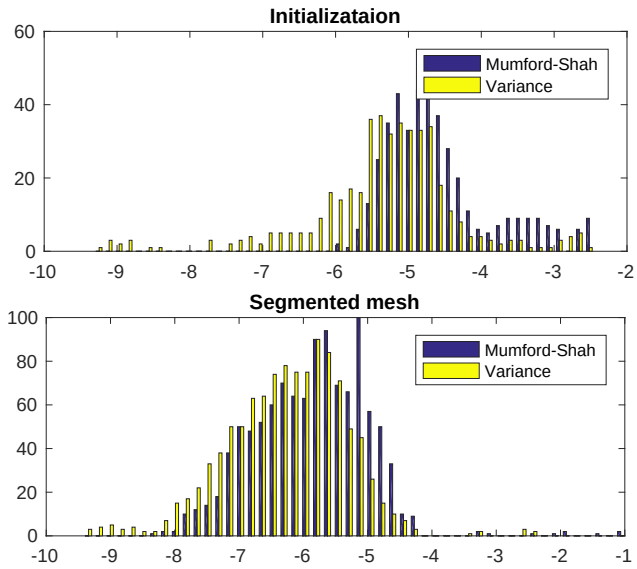


Figure 5.3: Log-Histogram of triangle energy in cement segmentation.

Triangle adaptation, described in Alg. 10, combines triangle splitting and triangle relabeling. Its purpose is to capture all regions of a phase, for example the holes in the letter 'B' in Fig. 1.5. For this we need to formulate a triangle energy which will trigger a split where we expect a subsequent energy minimization by triangle relabeling. Since adaptation process is not directly a part of minimization, we may choose a triangle energy which gives good results in practice.

Mumford-Shah energy would not be a good choice for this purpose. For example, a triangle with high Mumford-Shah energy may be homogeneous, but splitting such triangle would not lead to subsequent relabeling. Instead, we use the variance of the intensity inside the triangle as an energy function to trigger relabeling or split. This is computed as

$$E_t = \frac{1}{\text{area}(t)} \int_t (g - g_t)^2 \mathrm{d}\Omega \,, \tag{5.11}$$

where g_t is the mean intensity inside t. The energy threshold is α_t, and we split a triangle by inserting a new vertex at the midpoint of the triangle.

In our implementation, we calculate the integral over a triangle by discretizing its area into triangles covering one pixel.

Algorithm 10: Triangle adaptation

```
1 forall the triangle t do
2 |   Compute energy E_t of t                          // Eq. (Eq. 5.11)
3 |   if E_t < α_t then
4 |   |   Relabel t
5 |   else if t is stable and area(t) > MIN_EDGE^2/2 then
6 |   |   Split t
```

5.5.2 Interface edge adaptation

Edge adaptation includes splitting and collapsing stable interface edges, and is described in Alg. 11. As with the triangles, splitting the edge will not directly minimize the Mumford-Shah energy. Instead, energy is minimized in subsequent steps by vertex displacement.

To ensure that we split the edges which will subsequently move, we consider the forces derived in Eq. 5.8. For a stable edge, force on the two vertices is zero or

very small, but the force along the edge might still be considerable. Therefor, we trigger the edge splitting if the line integral

$$E_e = \frac{1}{L} \int_0^L \|f_{ij}^{\text{edge}}\| \mathrm{d}l \,, \tag{5.12}$$

of external forces from (Eq. 5.8) is larger than the energy threshold α_e. The edge is split by inserting a new vertex at its barycenter.

An edge is collapsed if a pair of interface edges has low curvature. The feature is a part of DSC and is performed by merging endpoints of an edge. Junctions vertices (vertices that separate more than two regions) will not be collapsed.

Algorithm 11: Edge adaptation

```
1 forall the interface edge e do
2 |   Compute energy E_e of e                    // Eq. (Eq. 5.12)
3 |   if e is stable then
4 |   |   if E_e > α_e and length(e) > MIN_EDGE then
5 |   |   |   Split e
6 |   |   else if low curvature then
7 |   |   |   Collapse e
```

5.5.3 Mesh thinning

After subdividing the mesh and capturing small regions, we may have redundant vertices. The purpose of the thinning process is to clean up these vertices while maintaining the quality of the mesh. Thinning the mesh includes vertex removal and needle triangle removal.

To avoid changing the interface, we only remove the interior vertices, i.e. vertices which are not a part of the interface. An interior vertex is removed if its one ring (neighbor triangles) is homogeneous. We use the intensity variance (Eq. 5.11) as the criterion of homogeneity. We remove the vertex by merging it to one of its neighbor vertices.

Vertex removal may leave us with needle (elongated) triangles, especially between regions of different mesh resolution. To insure mesh quality we removing needle triangles. Due to adaptive resolution, the quality of the mesh is defined in terms of triangle angles. Removal method is shown in Fig. 5.4: we collapse two long-edges (e_1 and e_2) and add a new vertex to the center of the needle-triangle. In order to maintain the interface, we do not collapse the needle triangles adjacent to interface.

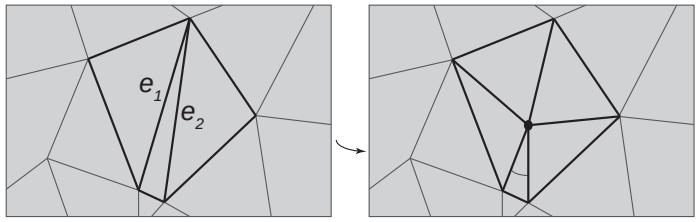
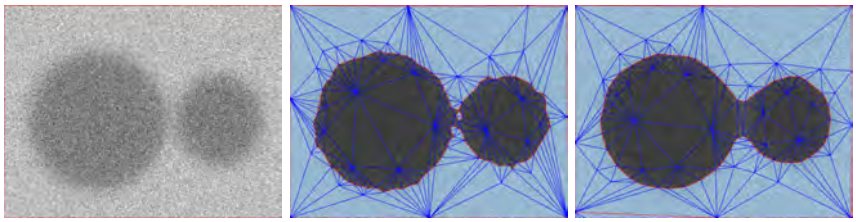


Figure 5.4: Removing a needle triangle

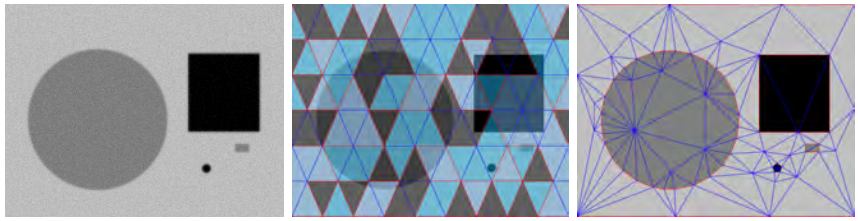
5.6 The parameters

Four parameters are important for our algorithm: the threshold for triangle adaptation and relabeling α_t, the threshold for edge splitting α_e, the minimal edge length MIN_EDGE, and the curve length parameter from Mumford-Shah functional α.

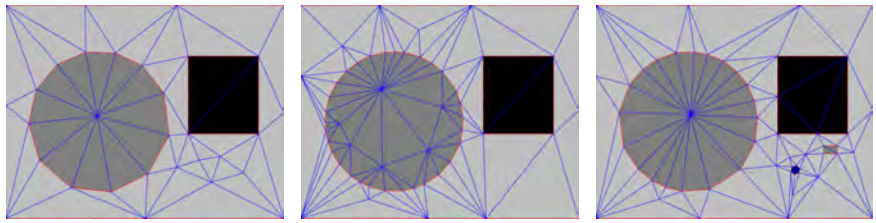


(a) Input image **(b)** $\alpha = 0.1$ **(c)** $\alpha = 1$

Figure 5.5: Effect of the smoothness weight α



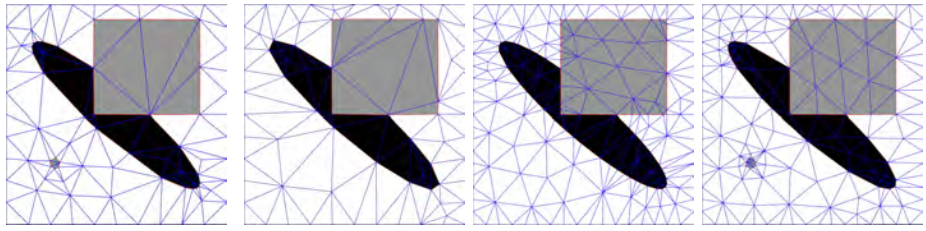
(a) Input image **(b)** Initialization **(c)** Reference



(d) Increasing MIN_EDGE **(e)** Increasing α_t **(f)** Increasing α_e

Figure 5.6: Effect of changing mesh parameters. (a) Test image of size 800×600 pixels. (b) Random initialization. (c) A reference segmentation using parameters MIN_EDGE $= 10$, $\alpha_t = 0.003$ and $\alpha_e = 10$. (d) MIN_EDGE $= 100$. (e) $\alpha_t = 0.01$. (f) $\alpha_e = 30$.

The parameter α from (Eq. 2.11) depends on the modelling need. We used values from 0.01 to 1 depending on the noise level. With large α, the segmentation is less sensitive to noise but may not separate all regions or oversmooth the boundary (Fig. 5.5 c). Besides, the captured areas will be smaller (Fig. 5.18). Generally, we set α to 0.1.



(a) Reference **(b)** larger edge **(c)** larger α_t **(d)** larger α_e

Figure 5.7: Effect of changing mesh parameters

The influence of other three parameters is shown in Fig. 5.6 and Fig. 5.7. The

minimal edge length MIN_EDGE should be set by estimating the size of the smallest region we want to segment and the desired smoothness of the segmentation boundary. As shown in 5.6d, the small circle and square are not segmented with large MIN_EDGE, and the big circle is represented with fewer line segments.

The triangle energy threshold α_t allows us to control the level of detail we want to segment. As we start with a sparse mesh, we need triangle subdivision to capture the small regions. Large α_t makes it more difficult to trigger triangle splitting, so small details will be lost, see 5.6e. However, the smoothness of the segmentation boundary is not influenced by α_t.

The edge energy threshold α_e defines the resolution of the curves. In 5.6f, a larger threshold leads to fewer line segments on the big circle. However, the small object are still segmented.

To estimate reasonable values for thresholds α_t and α_e we can sample a homogeneous area on the image: α_t can be set by measuring the energy of a small and homogeneous triangle, while α_e can be set by measuring the energy of an edge, which has been aligned to the boundary of the regions in the image g.

Parameter N_a in Alg. 8 controls how many iterations elapses between two mesh adaptation steps. This will not influence the final result, but only the total segmentation time. To estimate the optimal N_a, we measure the time to convergence with N_a in range from 1 to 20. The results in Fig. 5.8 indicate that N_a should be 3 to 5.

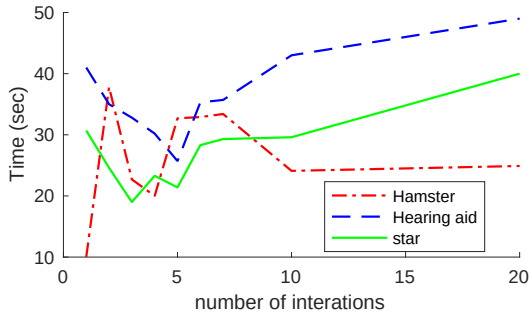
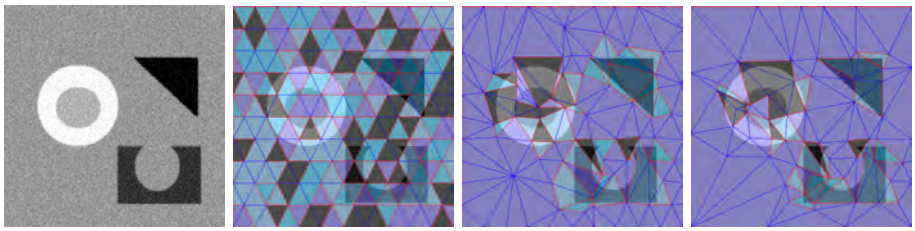
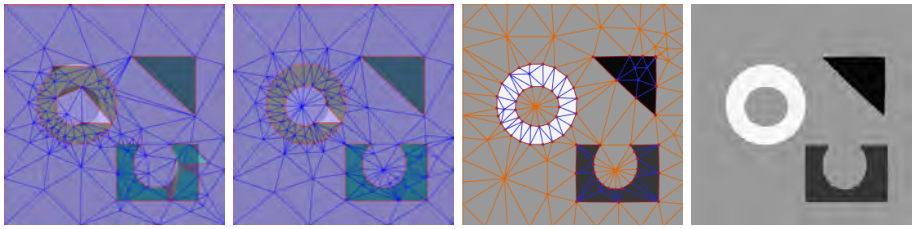


Figure 5.8: Time to convergence depending on number of iterations between two adaptation steps.



(a) Input image **(b)** 4-phase initialization **(c)** Iter 25 **(d)** Iter 30



(e) Iter 75 **(f)** Iter 100 **(g)** Output mesh **(h)** Output image

Figure 5.9: Segmentation of a four-phase image

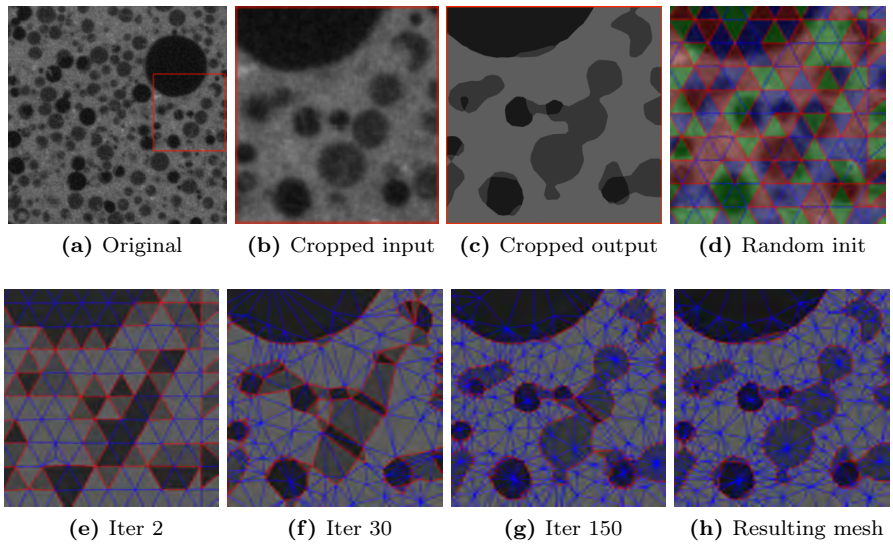
5.7 Results and discussion

To show our results on challenging geometry we segment a set of synthetic images, including a generated geometry of a fuel cell. To demonstrate the further properties of our method we segment slices from CT scans: a hamster toy, hearing aid device, cement sample and dental implant. For all experiments, if an initialization of label is not shown, the initialization is random. We also discuss the performance and convergence of our method in the end of this section.

5.7.1 Properties

Natural multi-phase support Fig. 5.17 shows various examples of multi-phase segmentation. Our method represents multiple phases on a single triangle mesh. In contrast, level set method either uses multiple level set functions [163] or requires interpretation from one level set function [96].

Adaptive mesh for optimal representation Fig. 5.11 shows the segmentation of a large object with small features. Adaptive mesh minimizes mesh

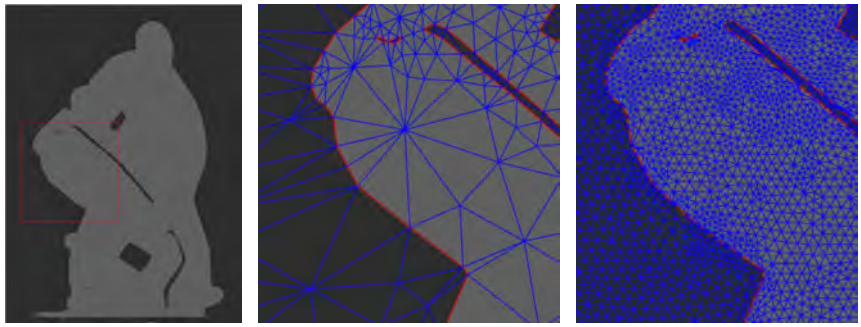


(a) Original **(b)** Cropped input **(c)** Cropped output **(d)** Random init

(e) Iter 2 **(f)** Iter 30 **(g)** Iter 150 **(h)** Resulting mesh

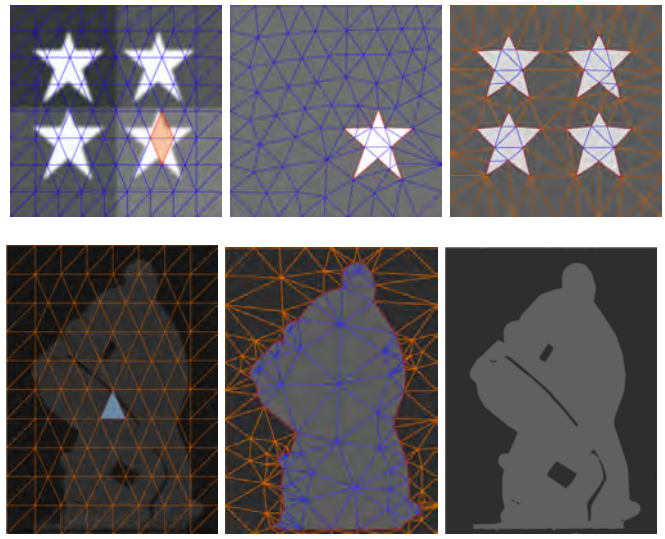
Figure 5.10: Evolution of mesh during segmentation of a gray scale image (250×250) showing cement with air and polymer bubbles. The mesh are shown in the cropped area size 90×90.

size but still captures the small features and maintains the quality of the mesh. 5.11c shows comparable segmentation using a uniform mesh, with the number of triangles is 20 times higher than that of the adaptive mesh.



(a) Segmentation **(b)** Adaptive image **(c)** Dense mesh

Figure 5.11: An adaptive mesh and a dense mesh achieving comparable segmentation but with different resolutions. Image size 440×600; 5.11(b) $\sim$ 1700 triangles; 5.11(c) $\sim$ 20300 triangles

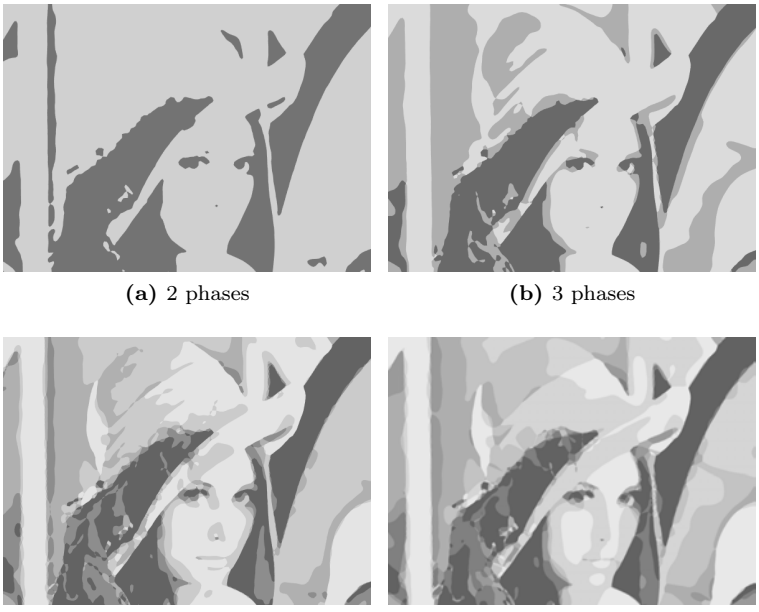


(a) Initialization **(b)** Relabel off **(c)** Relabel on

Figure 5.12: Segmenting a single object and multiple objects. Both segmentations use the same initialization

Full topology control: Triangle relabeling is necessary for minimizing Mumford-Shah potential. However, for a different segmentation purpose, we might want

to capture only a single region. We demonstrate how this is achievable using DSC in Fig. 5.12. Without relabeling, a single object is segmented, and all other objects are ignored.



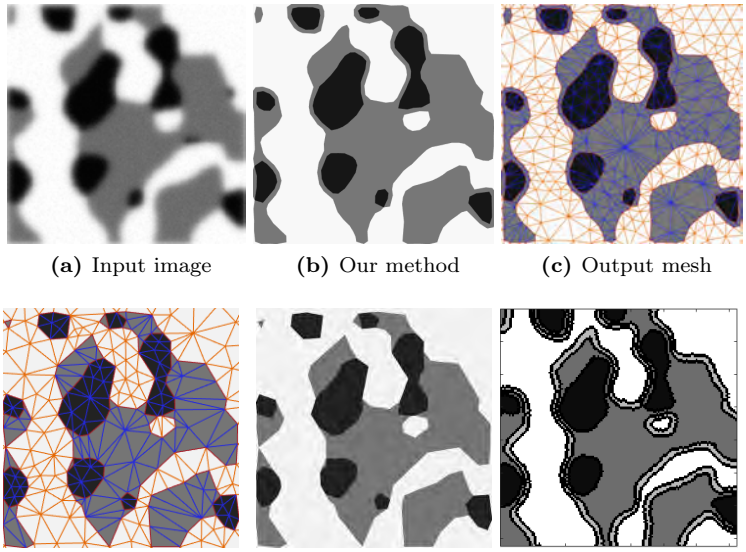
(a) 2 phases **(b)** 3 phases

(c) 5 phases **(d)** 7 phases

Figure 5.13: Segmentation of the red channel of the Lena image (512×360) with mininal edge length $\epsilon = 2$ pixels

Controlling the number of phases Our method allows us to define the exact number of phases we want to segment, which we demonstrate on a fuel cell image. In Fig. 5.14 (b-e) we show a segmentation in three phases using our method. With the level set method [163], implemented in [169], the three-phase segmentation using $2^2 = 4$ states results in an additional fourth phase, as shown in 5.14f.

Blur boundary Still considering Fig. 5.14 (b, c), the blurred boundaries between the black and the white phase are captured as gray. This minimizes Mumford-Shah energy (also seen in 5.14f), but may be undesirable for some applications.



(a) Input image **(b)** Our method **(c)** Output mesh

(d) Larger MIN_EDGE **(e)** Larger MIN_EDGE **(f)** Level set method

Figure 5.14: Segmentation of synthetic fuel cells with noise and blur.

With our method, this issue may be reduced by controlling the minimal edge length. In Fig. 5.14 (d, e) we show the results after when we increasing MIN_EDEGE. We remove the undesirable blur boundary, but we loose some small regions and deteriorate the curve smoothness.

Smoother curves and reduced noise In Fig. 5.15 we provide further comparison with results obtained using an implicit curve. Our results show smoother curves and correct capturing. In the segmentation of star image (a) our method segments the four phases, whereas level set method could not identity the dark gray phase. The reason is that level set function is more sensitive to noise and has to be accompanied with a larger smooth coefficient. In the segmentation of scan of a hearing aid device (b) level set method captures noise in the background. In the segmentation of the synthetic fuel cells (c) our method shows smoother interface and does not segment the unwanted phase. In the segmentation of dental implant scan (d) neither method could capture the air (dark and small regions) correctly. In the segmentation of cement scan (e) level set labels wrong phases to some gray circles.

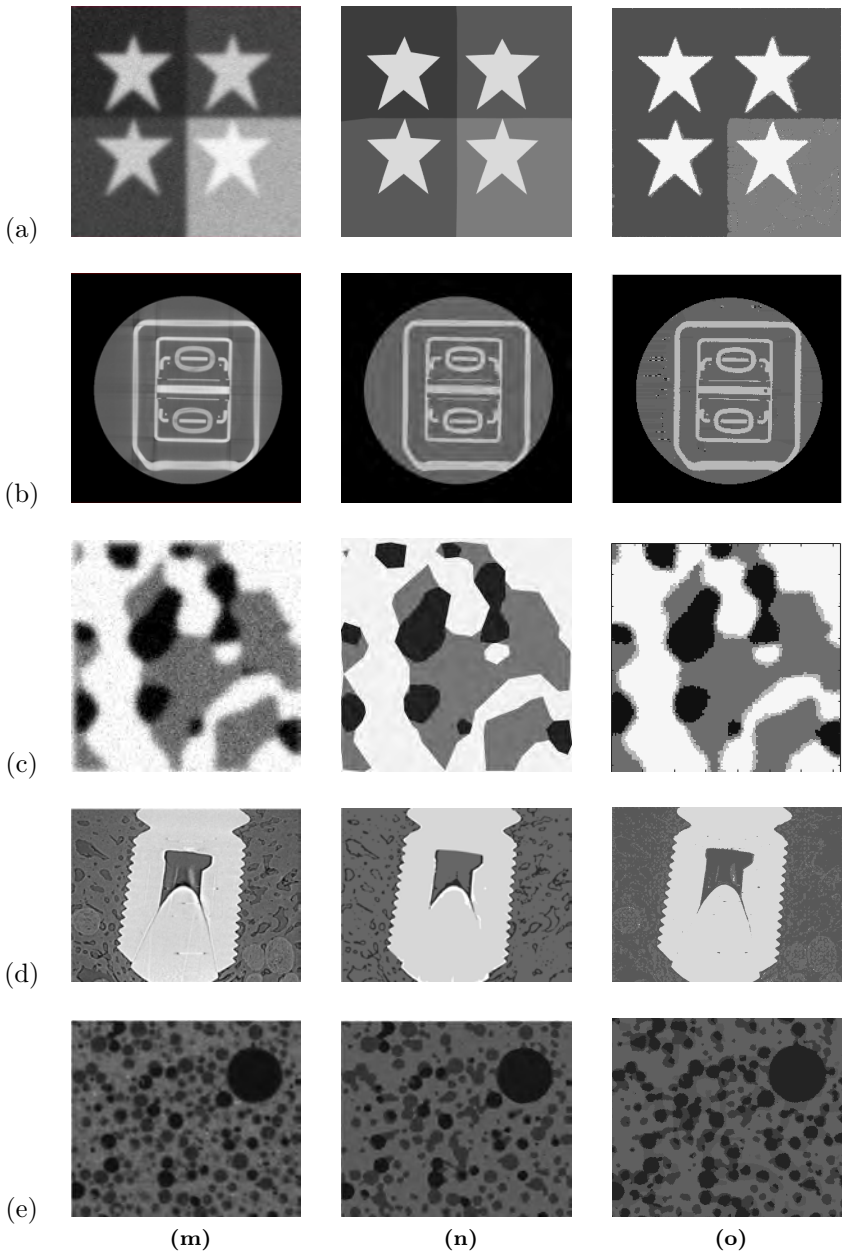


Figure 5.15: Comparing our results with segmentation using implicit curve. *First row*: Input images. *Second row*: Our method. *Third row*: Level set method [163]. (a) Four phases (b) A hearing aid device (c) Fuel cell (d) Dental implant (e) Cement

5.7.2 Performance

Convergence Fig. 5.16 (a) compares convergence of our method with the level set method. Methods perform comparably in the beginning, but our method converges to the solution faster when the energy becomes smaller. The fluctuations occur in the convergence plot because we adapt the mesh once every 20 iterations. These fluctuations are actually small, as shown in , but were scaled by semilog plot.

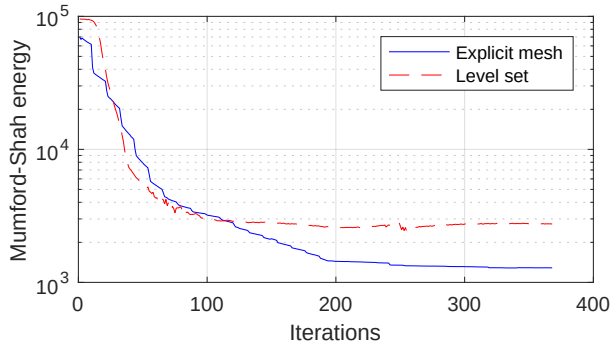


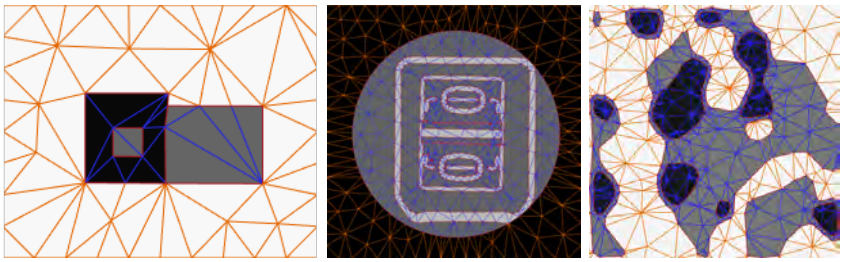
Figure 5.16: Mumford-Shah energy with respect to number of iterations in segmentation of scan of a hearing aid device. The input image of the hearing aid is in Fig. 5.15.

Computation time In Tab. 5.2 we provide the computation times of 9 examples in Fig. 5.17. Our code is implemented in C++, and level set method [163] is implemented in Matlab [169]. All experiments run on one core of a CPU 2.5 GHz, RAM 16 GB.

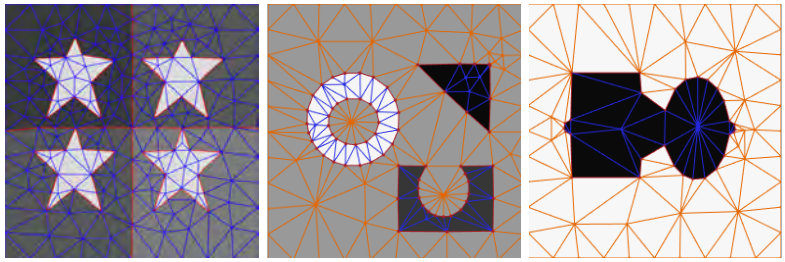
We could not set the same stopping criteria for the two methods. The reason is the inconsistency in weighting the curve length contribution to (Eq. 2.11). Instead, we stop the segmentation when the residual errors are stable.

Table 5.1: Parameters choices for segmentation of images in Fig. 5.17. Numbers reported under mesh size are number of vertices / number of faces.

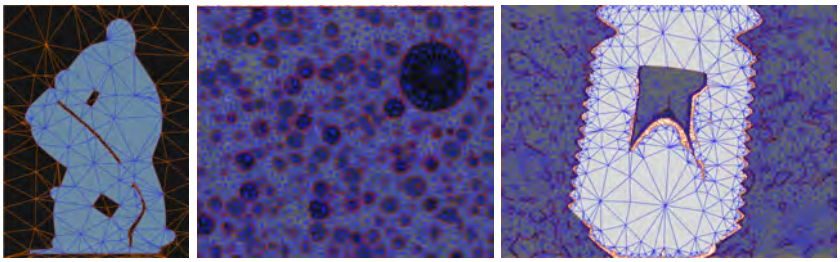
Fig. 5.17	Mesh parameters					Level sets
	α	α_T	α_e	MIN_EDGE	Mesh size	α
(a)	0.1	0.2	2	10.0	59 / 80	0.3
(b)	0.1	0.08	3	3.0	607 / 1128	0.1
(c)	0.1	0.08	1	3.0	480 / 880	0.1
(d)	0.1	0.08	1	2.0	189 / 340	0.01
(e)	0.1	0.2	2	15.0	90 / 138	0.1
(f)	0.1	0.2	2	5.0	144 / 186	0.1
(g)	0.1	0.2	0.3	3.0	89 / 136	1
(h)	0.2	0.08	10	4.0	3098 / 6110	0.071
(i)	0.2	0.5	5	8.0	4639 / 9192	0.05



(a) Square **(b)** Hearing aid **(c)** Fuel cell



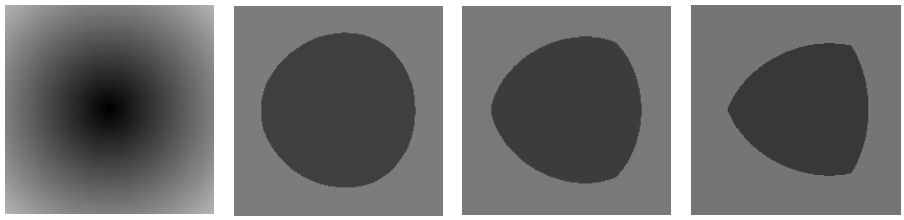
(d) Stars **(e)** Four phases **(f)** Single object



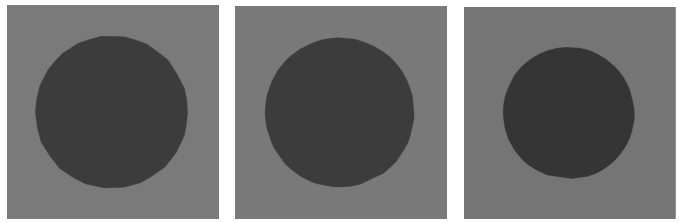
(g) Hamster **(h)** Cement **(i)** Dental implant

Figure 5.17: Segmented mesh of samples for performance measurement.

Fig. 5.18 shows the effect of the coefficient α to the curve length. In level set the curve length is measured by $\int_\Omega |\nabla H(\phi)| \mathrm{d}\Omega$. We can see that there is a big difference in the curve length segmented with the same α for two methods. In 5.18c and 5.18d, the segmentation does not maintain a round shape due to numerical error, as level set function suffers from numerical diffusion. Other parameters can be seen in Tab. 5.1.



(a) Input image **(b)** 0.01 / 550 **(c)** 0.03 / 522 **(d)** 0.05 / 479



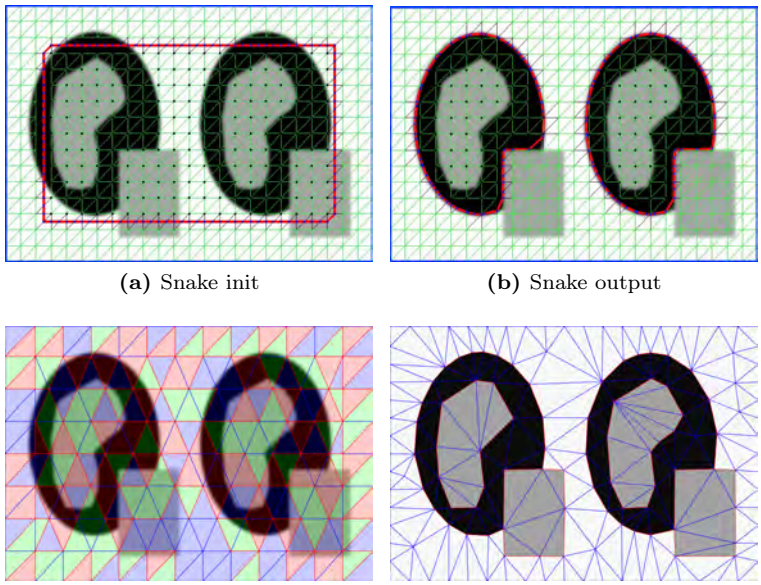
(e) 0.01 / 448 **(f)** 0.1 / 440 **(g)** 1 / 389

Figure 5.18: Effect of coefficient α to curve length. (a) Radial gradient circle. (b-d) Level set method; (e-g) Our method Numbers reported under images are α / Length(Γ).

The results show that the time complexity of our method depends on the resolutions of the mesh, which reflects the complexity of the regions we want to segment. Time complexity of the level set method depends on the resolutions of the images consistently. Generally, our proposed method is up to 20 times faster than level set method (for images with size around 1000 pixels). Even though C++ is often more efficient than MATLAB, these numbers still show that our method is comparable or faster than level set method in term of performance. We also need fewer iterations for the mesh to converge to the solution. Generally, the number of iterations of level set method is three times higher than that of our method. The hamster image took most iterations due to the thin structures which required many adaptive subdivision steps.

5.8 Comparison with other methods

Comparison with the T-snakes method Fig. 5.19 demonstrates a snakes segmentation. Our output is a three-phase segmentation, and the initialization of triangle labels is random. As we do not rely on a fixed grid, our method can segment the image with a coarser adaptive mesh. Compared with our method, the t-snakes [109] has several limitations: curves stick at local minimum and are heavily dependent on initialization; less accuracy in capturing sharp corners due to the vertex-based force; no adaptive resolution mesh; no easy multi-phase support; and accuracy depends on the size of the underlying grid.

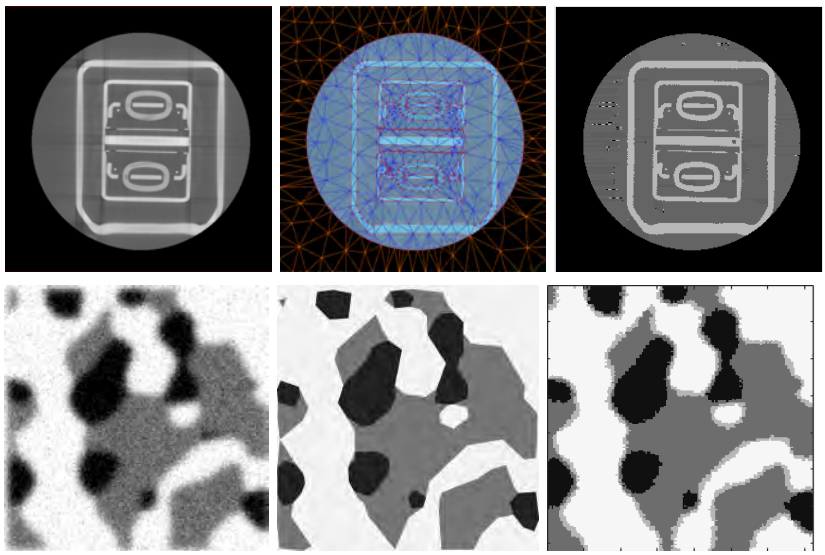


(a) Snake init **(b)** Snake output

(c) Our random initialization **(d)** Out output

Figure 5.19: Comparison between t-snakes and our method.

Comparison with level set method To compare our method with level set method, we used a Matlab implementation [169] capable of segmenting up to 4 phases represented by two level sets. The results in Fig. 5.20 shows that our method is less sensitive to noise as we distinguish between mesh resolution and image resolution. Besides, we have advantage on controlling the exact number of phases and handling blur boundary (by controlling the edge length ϵ).



(a) Original image **(b)** Our segmentation **(c)** Level set

Figure 5.20: Comparison between level set method and our method. Top row: scan of a hearing aid device (500×500). Bottom row: fuel cell scan (350×350)

Table 5.2: Computation time comparison for segmentation of images in Fig. 5.17.

Fig. 5.17 image	**size**	**Level sets**		**Ours**	
		iters	time (s)	iters	time (s)
(a) Square	1280×1024	250	273.6	80	10.8
(b) Hearing aid	1024×1024	500	578	150	25
(c) Fuel cell	200×200	400	3.6	100	1.5
(d) Hamster	340×460	150	15.5	350	6
(e) Stars	500×500	400	64	160	4.5
(f) Four phases	700×700	500	185	160	9
(g) Single object	500×500	100	17.2	40	1.5
(h) Cement	350×350	450	19.2	140	10
(i) Dental	1100×850	400	288	180	60

Tab. 5.3 provides the performance comparison for five examples. All experiments run on one core of a CPU 2.5 GHz, RAM 16 GB. We could not set the same stopping criteria for the two methods because of the inconsistency in weighting the curve length contribution in (Eq. 2.11). Instead, we stop the segmentation

Table 5.3: Computation time comparison for segmentation of images in Fig. Fig. 5.17.

Fig. Fig. 5.17 image	size	Level sets		Ours	
		iters	time (s)	iters	time (s)
(a) Square	1280×1024	250	273.6	80	10.8
(b) Hearing aid	1024×1024	500	578	150	25
(c) Fuel cell	200×200	400	3.6	100	1.5
(d) Hamster	340×460	150	15.5	350	6
(e) Stars	500×500	400	64	160	4.5
(f) Four phases	700×700	500	185	160	9
(g) Single object	500×500	100	17.2	40	1.5
(h) Cement	350×350	450	19.2	140	10
(i) Dental	1100×850	400	288	180	60

when the residual errors are stable.

The results show that the time complexity of our method depends on the resolutions of the mesh, which reflects the complexity of the regions we want to segment. Time complexity of the level set method depends on the resolutions of the images consistently. Generally, our proposed method is up to 20 times faster than level set method. Even though C++ is often more efficient than Matlab, these numbers still show that our method is comparable or faster than level set method.

5.9 Conclusion

This paper proposes an algorithm for multi-phase image segmentation using a deformable mesh. We represent a piecewise constant function in the image domain using a triangle mesh and segment the image by minimizing the Mumford-Shah functional. Furthermore, an adaptive mesh algorithm is proposed to optimize the resolution of the triangle mesh.

The proposed method has several advantages. We can segment arbitrary number of phases with one triangle mesh. In comparison with the level set method, our method is less sensitive to noise and is faster and generally requires fewer iterations. On the other hand, our method does, however, require the user to select four control parameters. Fortunately, these parameters are also useful for controlling a desired output.

Our method inherits the properties of explicit methods: the advantages in having an explicit geometry, which allows easy computations of geometric descriptors. While segmentation with an explicit mesh would often preclude topology changes, this is not true in our case, since we rely on the Deformable Simplicial Complex method.

In future work, we will extend the method to 3D and consider automatically deriving control parameters. Besides, an implementation of the 2D segmentation will be made available to public.

Chapter 6

3D Intensity-based image segmentation

Tuan T. Nguyen, Vedrana A. Dahl, J. Andreas Bærentzen,
Camilla H. Trinderup
Technical University of Denmark

Published to 29^{th} *British Machine Vision conference (BMVC)*

Abstract In life science and material science, it is often desirable to segment a volumetric data set in such a way that multiple materials (phases) are segmented and a tetrahedral mesh representation is obtained for each segment for downstream applications. Unfortunately, obtaining a mesh, typically from CT or MRI scan, is challenging, especially in 3D. This paper proposes a novel approach for volume segmentation using a tetrahedral mesh. Our method employs a deformable model that minimizes the Mumford-Shah energy function. We apply our method to several CT data sets in order to demonstrate its advantages: multi-phase support, robustness to noise, and adaptive resolution outputs. Our method is based on the Deformable Simplicial Complex (DSC) method for tracking deformable interfaces which is designed specifically to deal with topology changes.

6.1 Introduction

In recent years, it has become an increasingly important concern to build simulation meshes from data acquired using one of the many CT or MRI based scanning modalities. Generally, tetrahedral meshes for finite element or geometric analysis are required, and very often the objects being scanned are heterogeneous, leading to the need for a multi-phase segmentation of the scanned object.

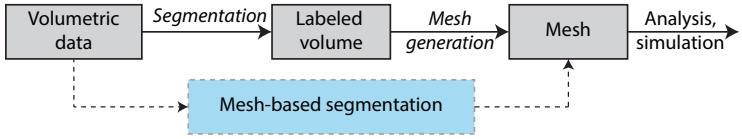


Figure 6.1: Common work flow in volumetric segmentation for material analysis

An example we use in this paper is concerned with segmenting three phases in the scan of a solid oxide fuel cell. Characterization of the fuel cell, and simulation of its operation, requires a precise description of interfaces between materials, including the triple phase boundary. Another example is segmentation of fibre bundles in composite material which can be used for estimating the material's overall stiffness and strength. The most common workflow for such analysis is illustrated in Fig. 6.1. The process is split into two parts: a segmentation that operates at the voxel level and mesh generation which produces the desired result from the segmentation.

In this paper, we propose to go directly from the initial volumetric data to a mesh-based representation of the segmentation as illustrated with the blue box in Fig. 6.1. The pivotal idea is to represent the entire domain as an irregular tetrahedral mesh where each tetrahedron is assigned a label corresponding to the material (or phase) to which it belongs. This simplifies dealing with multiple interfaces since the interface between two materials is simply the set of triangular faces shared by tetrahedra with corresponding labels. Moreover, we surmise that for further processing, typically simulation, it is a significant advantage that the internal representation is also the desired output representation.

Related ideas for two-phase segmentation are well researched in the literature named *deformable model*, a model that is strong against noise and artifacts and suitable for data with homogeneous regions representing materials [172]. First introduced by Terzopoulos *et al.* [156, 83, 157] this has become one of the most successful approaches to image segmentation. In deformable model, we are concerned with the curves/surfaces representation and the force model.

The force models were proposed initially [83, 157, 159, 38, 39, 171, 54, 99]. Despite the variety, all those models use local differential properties of image edges, hence the segmentation often sticks to local minima. User-driven forces may be required to achieve desired segmentation [83]. To overcome the above issue, many authors utilize a global energy function proposed by Mumford and Shah [118], one of the most popular model in deformable model with many applications [8]. This paper shall focus on deformable models that minimize the Mumford-Shah energy functional.

Minimizing the Mumford-Shah functional using an implicit representation (*e.g.* level set method) is popular due to the ease with which topology changes (*e.g.* splitting and merging) are handled. Deformable models using an implicit representation is based on curve evolution [126, 148, 149]. Solving it has been studied in depth with many proposals [13, 28, 29, 14, 4, 150, 86, 107, 31, 163, 140, 161]. Perhaps, the most popular model is the active contour without edge, a two-phase segmentation [31]. For multi-phase, implicit representations have problems in phase overlapping [183]; or are limited to fixed number of phases [163]; or suffer from ill-conditioned equations [96].

In contrast to implicit representations, explicit representations have advantage in representing multi-phase as the interfaces are literally defined. However, the difficulty in handling topological changes is a significant obstacle. Perhaps an effortless approach to overcome this issue is to borrow techniques from explicit interface tracking researches, which leads to many proposals to deal with topological changes. [180] explicitly resolve each intersection, [167, 108] generate new mesh based on an underlying fixed grid, and [33] utilize element deletion technique. Unfortunately, these methods do not support multi-phase yet. For multi-phase, [45] use collision technique, and [133] use Delaunay mesh generation. These two methods only maintain a surface triangle mesh. [110] utilize a tetrahedral mesh and detect topological changes using neighbor information. Among all of these methods, [108, 133] have been applied to image segmentation, but their force models are local. Very few researches accommodate the Mumford-Shah energy function with 3D explicit mesh, and they are limited to segmenting a single region *e.g.* [58].

In [119] the authors have utilized the interface tracking technique in [110] to minimize the Mumford-Shah energy function for 2D problem and showed improvement in accuracy compared to other deformable models. The current paper extends the method to 3D and inherits the advantages: multi-phase support; higher accuracy; automatic resolve junction vertices; and output is an adaptive tetrahedral mesh. Beyond generalizing the method to 3D, the contributions in this paper include:

- a scheme for computation of triangle-based forces with Gaussian quadrature formula (Sec. 6.2.3),
- a generalization that enables the method to work with various input model beside intensity (Sec. 6.4),
- and a new mesh adaptation scheme, in which paramaters are geometric-based and can be invisible to user (Sec. 6.3).

6.2 Method

6.2.1 Method overview

Given an image $I : \Omega \to \mathbb{R}$, we want to find a piecewise constant function $u(\mathbf{x}) = c_i$ if $\mathbf{x} \in \Omega_i$ (where Ω_i denotes a disjoint phase of constant intensity c_i) that minimizes the Mumford-Shah functional

$$E(u) = \sum_{i=1}^{N} \int_{\Omega_i} (I - c_i)^2 \mathrm{d}\Omega + \alpha \|\partial u\| \tag{6.1}$$

where $\|\partial u\|$ denotes the area of the interfaces, α is the smoothing coefficient, and N is the number of phases.

We define a piecewise constant function on a tetrahedral mesh by labeling the tetrahedra with a labeling function $\{l_i\} : \{t_i\} \to \mathbb{Z}, l_i \in [1, N]$. Faces that have coboundary tetrahedra with different labels define the interface (Fig. 3.4). Edges and vertices on the interface are interface edges and interface vertices.

Alg. 12 shows the overview of our algorithm. In our method, the unknowns are the position of interface vertices $\{\mathbf{p}_i\}$, phase intensities $\{c_i\}$ and the labeling function $\{l_i\}$. The function we want to find is $u = u(\{\mathbf{p}_i\}, \{c_i\}, \{l_i\})$. We treat the minimization problem independently for these unknowns, leading to three minimization problems correspond to step 1, 2, 3 in Alg. 12

$$1) \quad \min_{\{c_i\}} E(u) \qquad 2) \quad \min_{\{\mathbf{p}_i\}} E(u) \qquad 3) \quad \min_{\{l_i\}} E(u) \tag{6.2}$$

Algorithm 12: Algorithm overview

Input: image I, initial mesh $\mathcal{M}$

Initialize labeling function $\{l_i\}$
while *vertex displacements are large* **do**
  1) Update the phase intensities $\{c_i\}$ /* Sec. 6.2.2 */
  2) Compute forces on boundary vertices /* Sec. 6.2.3 */
    Deform the mesh with the DSC
  3) Update the labeling function $\{l_i\}$ /* Sec. 6.2.2 */
  4) Adapt the mesh to multi-resolution /* Sec. 6.3 */

6.2.2 Minimize E with respect to $\{c_i\}$ and $\{l_i\}$

Phase intensities We find c_i by setting the partial derivative of E with respect to c_i equal to zero

$$\begin{aligned}\frac{\partial E}{\partial c_i} &= \frac{\partial}{\partial c_i}\int_{\Omega_i}(c_i - I)^2 \mathrm{d}\Omega \\ &= 2c_i\int_{\Omega_i} d\Omega - 2\int_{\Omega_i} I d\Omega\end{aligned} \tag{6.3}$$

meaning c_i is the mean intensity of the image in phase Ω_i.

$$\frac{\partial E}{\partial c_i} = 0 \implies c_i = \frac{\int_{\Omega_i} I d\Omega}{\text{Volume}(\Omega_i)} \tag{6.4}$$

Labeling function We find the optimal label of a tetrahedron by choosing the phase that minimizes the energy (Eq. 6.1) inside the tetrahedron. Note that changing the label also changes the interface, which must be included in computation of the internal energy. For external energy, we approximate the volume integral with Riemann sum and discretize the tetrahedron to a set of sampling points with size of one voxel.

6.2.3 Minimize E with respect to interface vertices

We move interface vertices on the gradient descent direction that minimize the energy. Consider an interface vertex with position $\mathbf{p}_i$, it will be displaced by $\delta\mathbf{p}_i = \nabla_{\mathbf{p}_i} E dt$, where $\mathrm{d}t$ is the time step. We separate the two terms in Eq. 6.1 to external energy $E^{\text{ext}} = \sum_{i=1}^{N}\int_{\Omega_i}(I - c_i)^2 \mathrm{d}\Omega$ and internal energy $E^{\text{int}} = \alpha\|\partial u\|$.

Gradient of the two energy functions are called external force $\mathbf{F}^{\text{ext}}$ and internal force $\mathbf{F}^{\text{int}}$, respectively.

Compute internal force: The area of the interface can be computed explicitly as sum of areas of all interface triangles. Gradient of the internal energy with respect to a vertex v_i is

$$\mathbf{F}_i^{\text{int}} = \frac{\partial}{\partial \mathbf{p}_i} E^{\text{int}} = \alpha \sum_{f \in \mathcal{N}_i} \frac{\partial}{\partial \mathbf{p}_i} \text{Area}(f) = \alpha \sum_{f \in \mathcal{N}_i} \|\mathbf{p}_1 - \mathbf{p}_2\| \, \mathbf{h} \tag{6.5}$$

where f denotes an interface triangle; $\mathcal{N}_i$ denotes the neighbor of v_i; $\mathbf{p}_1, \mathbf{p}_2$ are the postilions of the two other vertices in f; and $\mathbf{h}$ is the normalized height vector corresponds to $\mathbf{p}_i$ (See Fig. 6.2).

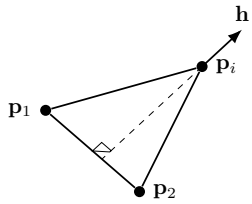


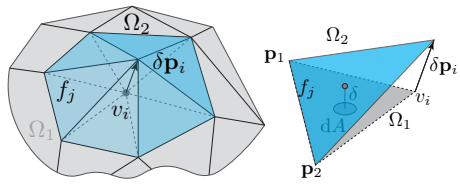
Figure 6.2: Derivative of triangle area with respect to a vertex

Compute external force: Local displacement of a vertex v_i only change the piecewise constant function u in the volume covered by the displacement of neighbor interface triangles (See Fig. 6.3a). Assume that the displace $\delta\mathbf{p}_i$ is small, image intensity on the interface triangles can be considered unchanged. Integration over the volume can be approximate with integration over the surface. External energy change caused by the displacement of v_i is

$$\Delta E_i^{\text{ext}} = \sum_{f \in \mathcal{N}_i} \int_f \left((I - c_1)^2 - (I - c_2)^2 \right) \delta \mathrm{d}A = \sum_{f \in \mathcal{N}_i} (c_1 - c_2) \int_f (2I - c_1 - c_2) \delta \mathrm{d}A \tag{6.6}$$

where δ denotes the orthogonal displacement of the small area $\mathrm{d}A$ (See Fig. 6.3b), and $\delta \mathrm{d}A$ represents for a small volume.

We approximate the integration with Gaussian quadrature formulas for triangles [44], which discretizes a triangle to a set sampling points with area coordinate $\{\xi_j, \eta_j, \zeta_j\}$. At a sampling point we have the image intensity I_j, and the orthogonal displacement is $\delta = \xi_j \delta\mathbf{p}_i \cdot \mathbf{n}_f$ ($\delta\mathbf{p}_i \cdot \mathbf{n}_f$ is the dot product that represents



(a) Vertex v_i moves by $\delta\mathbf{p}_i$ **(b)** Triangle integration

Figure 6.3: Illustration for computing forces on v_i. (a) Volume covered by triangles displacement changing intensity from c_2 to c_1 (b) Integration over volume covered by displacement of triangle f_j

the projection of $\delta\mathbf{p}_i$ on the normal direction $\mathbf{n}_f$ of triangle f, and ξ_j is the coordinate correspond to the vertex v_i). We obtain the energy change

$$\Delta E_i^{\text{ext}} = \sum_{f\in\mathcal{N}_i}(c_1-c_2)A_f \sum_{\text{point}j}\omega_j(2I_j-c_1-c_2)\xi_j\delta\mathbf{p}_i\cdot\mathbf{n}_f \tag{6.7}$$

where ω_j is the weight of the sampling point, and A_f is the triangle area.

In Eq. 6.7 we can see that $\delta\mathbf{p}_i$ only appears in the dot product $\delta\mathbf{p}_i\cdot\mathbf{n}$. Replace the gradient of a dot product: $\frac{\partial}{\partial\mathbf{p}}(\mathbf{p}\cdot\mathbf{n})=\mathbf{n}$, we obtain the final external force

$$\mathbf{F}_i^{\text{ext}} = \nabla_{\mathbf{p}_i}E^{\text{ext}} = \sum_{f\in\mathcal{N}_i}\left\{\mathbf{n}_f A_f(c_1-c_2)\sum_{\text{point}j}\omega_j\xi_j(2I_j-c_1-c_2)\right\} \tag{6.8}$$

6.2.4 Implementation

In our experiments, the image intensity is scaled to the range $[0:1]$. We transform image coordinate to continuous coordinates by linear interpolation. Our algorithm starts with an uniform mesh with average edge length ϵ. We utilize Otsu threshold method [127] to initialize the labeling function. In each iteration, we first compute the forces on interface vertices and then deform the mesh using the DSC method. In the second step, we execute adaptive resolution mesh algorithm in Sec. 6.3. This step can be performed once per 1-5 iterations. Finally, we stop the segmentation if vertex displacements are smaller than 0.01.

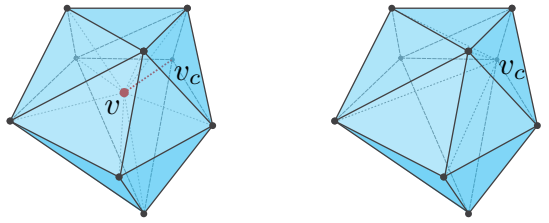
To boost the performance, we use sum table to compute $\{c_i\}$ and adaptive time step. Adaptive time step is applied for each interface vertex. We first estimate initial time step for all vertices so that maximum displacement is 0.4ϵ. In the following step, we scale the time step individually by 1.1 if a vertex moves in

the same direction as its previous displacement; and by 0.9 if it moves on the inverse direction. The time step is bounded at $[0.1 : 3]$. If a vertex is affected by topological events, we reset its time step. We utilize [122] to keep track of modified vertices.

6.3 Adaptive resolution mesh

Adaptive mesh is important not only for a compact representation but also for memory efficiency and performance in computation. There are two approach for adaptive mesh: subdivision and coarsening. Perhaps subdivision (start with a sparse mesh then locally subdivide where needed) is more intuitive, but it requires parameters for subdivision criteria. Though these parameters can be useful [119], tunning them can be difficult in 3D. In this paper, we follow the second approach that starts with a dense mesh then locally coarsen where needed.

We utilize edge collapse for mesh coarsening (See Fig. 6.4 for an illustration). The criteria for choosing collapsing edge is based on geometric information and in such a way that we deem user tweaking of parameters to be unnecessary. Our algorithm includes internal edge collapse and interface edge collapse described below.



(a) Edge (red) with vertices v, v_c **(b)** v is collapsed to v_c

Figure 6.4: Edge collapse

Internal edge collapse does not modify the interface. For each internal vertex v, we choose shortest edge neighboring to v as the potential collapsing edge. We utilize volume-length ratio to measure the tetrahedron quality [129] and only collapse if the qualities of new tetrahedra are larger than 0.3. This 0.3 threshold

is independent to input data set.

Algorithm 13: Coarsening tetrahedra

Input: mesh $\mathcal{M}$

```
foreach internal vertex v in M do
    e = potential collapsing edge
    v, v_c = two vertices of e
    if quality of new tets > 0.3 then
        Collapse v to v_c
```

Interface edge collapse modifies the interface, hence we only collapse an edge on flat surface (mean curvature at removing-vertex smaller than a 0.03). For each interface vertex v, we also pick shortest edge neighboring to v as the potential collapsing edge. We utilize triangle angles to measure triangle quality and only collapse if qualities of new interface triangles are larger than 10°.

Algorithm 14: Coarsening interfaces

Input: mesh $\mathcal{M}$

```
foreach interface vertex v in M do
    if interface is flat at v then
        e = potential collapsing edge
        v, v_c = two vertices of e
        if quality approved then
            Collapse v to v_c
```

6.4 Generalization

In many cases, intensity alone cannot distinguish features in an image. Methods like dictionary, filtering, *etc.* [87] are efficient approaches that output probability maps $\{\mathcal{P}_i : \Omega \to \mathbb{R}, i = 1 : N\}$ of voxels belong to phases. We apply our method for probability input by modifying the energy function to

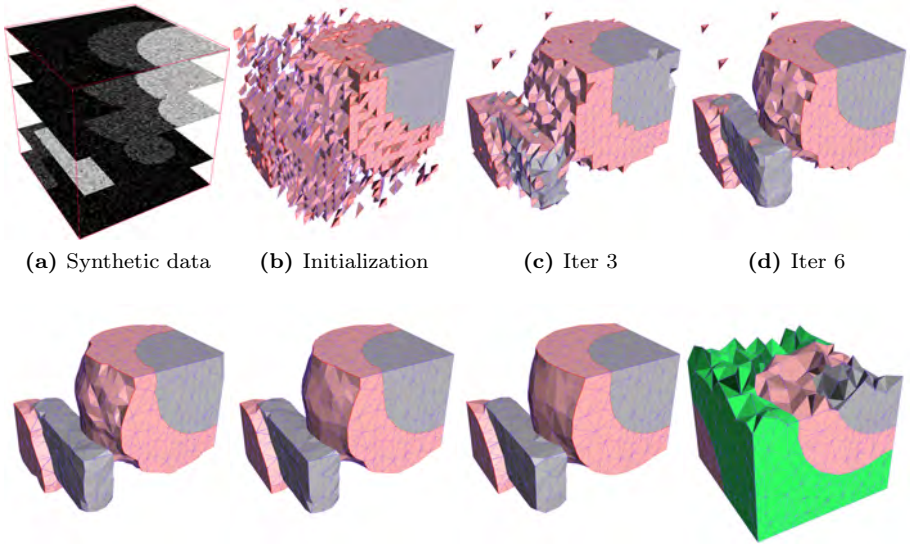
$$E^{\text{prob}}(u) = \sum_{i=1}^{N} \int_{\Omega_i} (1 - \mathcal{P}_i)^2 \mathrm{d}\Omega + \alpha \|\partial u\| \tag{6.9}$$

Minimizing E^{prob} is similar except that the external force in Eq. 6.8 now becomes

$$\mathbf{F}_i^{\text{ext}} = \sum_{f \in \mathcal{N}_i} \left\{ \mathbf{n}_f A_f \sum_{\text{point} j} \omega_j \xi_j (2 - \mathcal{P}_1 - \mathcal{P}_2)(\mathcal{P}_1 - \mathcal{P}_2) \right\} \tag{6.10}$$

6.5 Results and discussion

Accuracy Our method, and deformable models in general, is strong against noise. We tested our method on a three-phase synthetic data set with a variable level of Gaussian noise (See Fig. 6.5). In Tab. 6.1, one can see that our results are consistent while the noise level increases. In another example with CT data in Fig. 6.7, the result is visually accurate compared to a photograph of the real object.

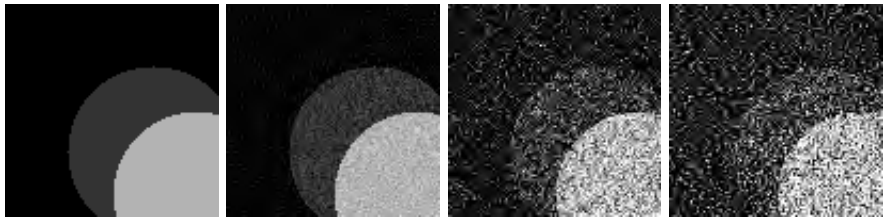


(a) Synthetic data **(b)** Initialization **(c)** Iter 3 **(d)** Iter 6

(e) Iter 10 **(f)** Iter 20 **(g)** Iter 50 **(h)** Cross section

Figure 6.5: Segmentation of synthetic data with Gaussian noise (mean 0 and variation 0.1)

Segmentation of synthetics data The synthetic data (Fig. 6.5a) is a three-phase image size $100 \times 100 \times 100$ with Gaussian noise. Fig. 6.5 demonstrates our tetrahedral mesh representing the segmentation. We also experiment with different noise levels (Fig. 6.6) and compare the segmentation with the ground truth image in Tab. 6.1.



(a) Original **(b)** 0.003 **(c)** 0.05 **(d)** 0.1

Figure 6.6: A slice of synthetic data with noise

Table 6.1: Experiment with noisy synthetic data.

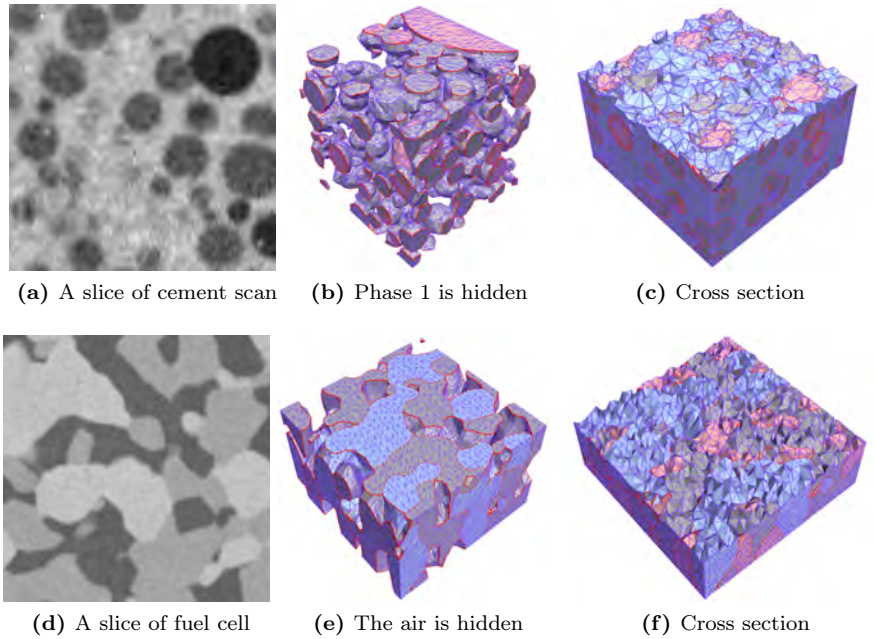
Experiment	1	2	3	4
Variance of the Gaussian noise (using `imnoise` in Matlab)	0.003	0.01	0.05	0.1
Mean squared difference between noisy volume and the ground truth	0.002	0.007	0.03	0.06
Mean squared difference between our result and the ground truth	0.0019	0.0018	0.0024	0.003
Correct segmented voxels	99%	99%	98.1%	97%

(a) Scan **(b)** Interface mesh **(c)** Output **(d)** Photo of the toy

Figure 6.7: Segmentation of scan of a plastic toy

Multi-phase support is the most important property of our method. We demonstrate the segmentation of two (Fig. 6.7), three (Fig. 6.8) and five (Fig. 6.9) phases, but the number of phases can be arbitrary. It is also noteworthy to men-

tion that the shared interfaces between phases are defined explicitly without fuzziness.



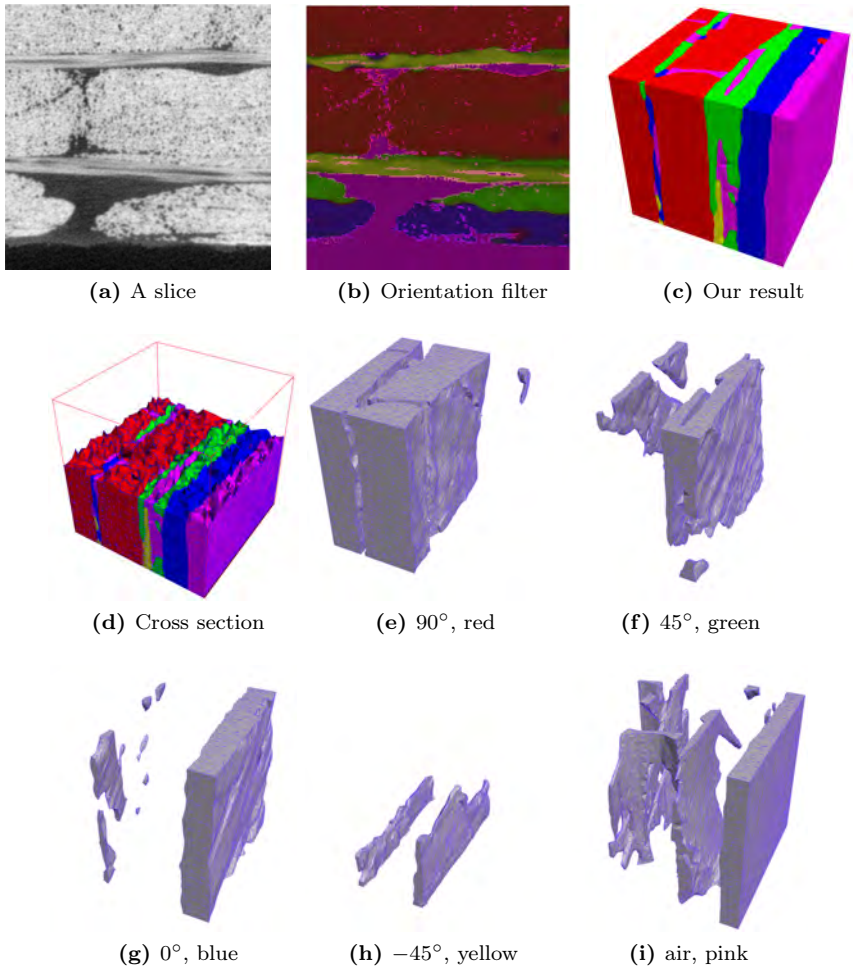
(a) A slice of cement scan **(b)** Phase 1 is hidden **(c)** Cross section

(d) A slice of fuel cell **(e)** The air is hidden **(f)** Cross section

Figure 6.8: Segmentation of cement scan and fuel cells scan. The third phase (the air) is hidden

Segmentation with probability input Fig. 6.9a shows a scan of composite material, where carbon fibers are grouped into bundles characterized by different orientations. By applying an orientation filter [87], we obtain five probability maps (Fig. 6.9b) of voxels belong to four bundle orientations or the resin. For this task our method takes probability input. The result of five-phase segmentation is shown in Fig. 6.9.

Comparison with 2D slice segmentation Using 3D information can improve the segmentation accuracy. In Fig. 6.10, it is difficult to segment the center slice unless we start with a good initialization that requires manual input. On the other hand, our method can still automatically segment that small region because we use support from other slices.



(a) A slice **(b)** Orientation filter **(c)** Our result

(d) Cross section **(e)** 90°, red **(f)** 45°, green

(g) 0°, blue **(h)** −45°, yellow **(i)** air, pink

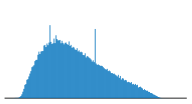
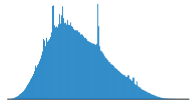
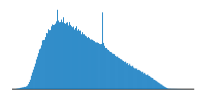
Figure 6.9: Fiber bundle segmentation, with fiber orientation filter.

(a) Test data surface (b) Center slice (c) with noise (d) Our segmentation

Figure 6.10: Segmentation contain slices with small features. Noise is Gaussian noise with variation 0.1. Image size $100 \times 100 \times 80$.

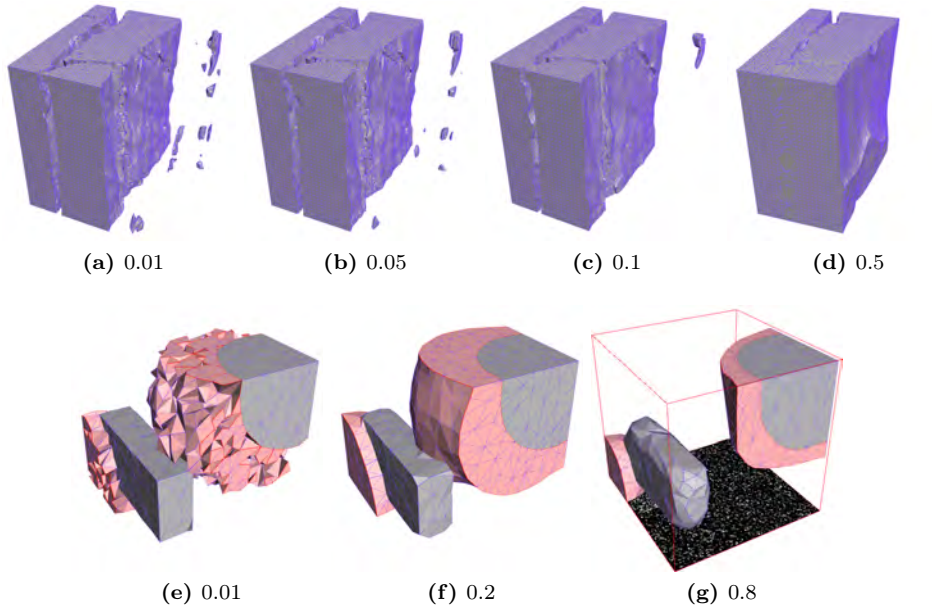
Output is an adaptive mesh is an advantage of our method over implicit representation. Adaptivity is an integral part of the algorithm, and it helps reduce the mesh size from initial mesh to final output up to 70% while still maintains a good quality mesh (mean dihedral angle is around 45° and smallest dihedral angle is acceptable in Tab. 6.2). This quality is sufficient for visualization and measurement. For procedures that require higher quality, we can tweak the threshold in the adaptivity algorithm and the DSC, which also leads to a less adaptive output.

Table 6.2: Mesh statistic. All numbers in (c,d) are in thousands. (a) min/max dihedral angle (b) Dihedral angle histogram [0° : 180°] (high peaks at 45° and 90° are tetrahedra at the boundary of the domain) (c) no. init tetrahedra / no. final tetrahedra / no. final interface traingles (d) no. voxels

	Fig. 6.7	Fig. 6.8a	Fig. 6.8c	Fig. 6.9
(a)	4.9° - 176°	3.4° - 179°	1.8° - 179°	2.8° - 178°
(b)				
(c)	317 / 92 / 9.4	257 / 220 / 61	260 / 140 / 27	421 / 185 / 31
(d)	5584	1000	2250	8000

Two parameters are important for our method: the edge length ϵ at initialization and the smoothing coefficient α. Choosing ϵ is straightforward as it represents the size of the smallest feature we want to segment. Choosing α requires some trial and error. Fig. 6.11 shows the effect of α: If α is too small, we segment noise while a large α tends to over-smooth. In our experience, the

best α is often in the range from 0.01 to 0.5 (Tab. 6.3).



(a) 0.01 **(b)** 0.05 **(c)** 0.1 **(d)** 0.5

(e) 0.01 **(f)** 0.2 **(g)** 0.8

Figure 6.11: Smoothing effect. Top: Phase 1 in Fig. 6.9. Bottom: Fig. 6.5

Table 6.3: Experiment detail. The computation time is separated to forces computation and meshing

Data set	Volume size	Edge length	α	# iters	Time [*sec*]	Energy (int/ext)
Fig. 6.7	$170 \times 150 \times 219$	5	0.03	50	64 - 114	11000 / 1000
Fig. 6.8a	$100 \times 100 \times 100$	3	0.01	60	91 - 205	7600 / 3900
Fig. 6.8c	$150 \times 150 \times 100$	4	0.03	28	59 - 74	3040 / 4983
Fig. 6.9	$200 \times 200 \times 200$	5	0.1	70	175 - 595	340000 / 25900

6.6 Conclusion

This paper proposes a volume segmentation method using a deformable model based on a tetrahedral mesh. Our method minimizes the Mumford-Shah functional, which is a global, noise resilient energy function.

Our method is based on DSC which is an explicit (or Lagrangian) method for

interface tracking that is still volumetric in the sense that the entire domain is partitioned into labeled tetrahedra. It is this approach which allows us to easily handle multiple materials while also resolving topological changes during segmentation.

Chapter 7

Deformable mesh evolved by similarity of image patches

Anders B. Dahl, Vedrana A. Dahl, Tuan T. Nguyen
Technical University of Denmark

To be submitted to *Asian Conference on Computer Vision (ACCV) 2018*

Abstract We propose a deformable model for manually initialized segmentation of images, which may contain both textured and non-textured regions. Image segments and segment boundaries are represented on a deformable triangle mesh, providing all advantages of an explicit geometry representation, but allowing for adaptive topology. Deformation forces are computed using a probabilistic model of local self-similarity, based on clustering of image patches. Both our curve representation and our similarity model naturally support multi-label segmentation. We demonstrate the properties of our approach on a number of natural color images as well as composed textured images.

7.1 Introduction

A deformable contour is a curve in the image domain which evolves under the influence of forces inferred from the image and from the curve itself. The approach is very popular in image segmentation, since tracking segment boundaries opens the possibility for regularization and for incorporation of shape priors. Proposed methods vary largely in two characteristics: the representation of the deformable curve and the derivation of the forces deforming the curve.

In general the curve representation falls in two groups, each with its advantages. An explicit curve (also called parametric curve, or snakes [83]) is represented as a sequence of points connected by line segments. Such a representation is straightforward, uses any desired resolution of points, and a curve deforms easy by displacing points. An implicit curve (largely based on level sets, e.g. [25]) defines an auxiliary function on the whole image domain, and the curve is located where the function changes sign. The most important advantage of an implicit curve is topological adaptivity.

Curve deforming forces are numerous. Local methods [84, 25, 105] attract curves to edges or other features characterizing segment boundaries. More global region-based forces [30, 174, 31] utilize characterization of segment regions providing a higher robustness to the method. In one of the foremost examples of the regional approach, active contours without edges by [30], each iteration of the curve deformation consists of two steps. First, the mean intensity of each segment is extracted from the current curve positions. Second, the curve is evolved depending on whether intensities under the curve lie closer to one or another extracted mean.

Our approach offers advances in both curve representation and in deforming forces. The mesh-based curve representation which we use is explicit, but it provides topological adaptivity. Further advantages of this representation are natural multi-label support, adaptive curve resolution and control of topological changes.

Our curve deforming forces are region-based and bear resemblance to active contours without edges. However, instead of averaging over pixel intensities we perform averaging over image patches, consequently encoding local self-similarity. The resulting forces are therefore able to segment any type of regions, characterized by both texture and intensity.

7.1.1 Related work

The most important limitation of the explicit curve representation originally proposed in [83] is the lack of topological adaptivity. Providing an explicit curve with topological adaptivity requires resolving curve intersections, examples include both 2D [108, 109] and 3D [167, 181] representations. This may be greatly simplified by using a deformable triangle mesh [132, 110]. Each triangle in the mesh is given a label that indicates to which segment it belongs. The curve (segment boundary) consists of the edges shared by triangles that have different labels.

Applications of deformable meshes in volume and image segmentation include [58] and [48], both with meshes of uniform resolution. The resolution adaptivity is introduced in [5] for segmentation of intensity-based regions. In this paper we extend upon [5] by incorporating forces based on self-similarity, allowing us to segment textured regions.

The typical approach to texture segmentation involves mapping the image to a texture descriptor space. Here the assumption is that descriptors within textures are similar while they differ between textures. Such an approach was suggested by [31] using texture orientation, which has been extended in e.g. [137] using the structure tensor and level sets. For better performance, the scale of the structure tensor is automatically estimated in [20, 21], while [19] utilizes diffusion.

Many other texture descriptors characterizing the local image structure have been suggested. These include local fractal features [162], gradient histograms [50, 142], local binary patterns [124], textons [104], and more. Images often contain texture on different scale that can be deformed or rotated versions of the same texture. Typically, this is handled in by designing descriptors invariant to such properties.

A related approach for image segmentation is based on sparse dictionaries of image patches [59, 101] where a dedicated dictionary is built for each texture class. Similar methods focusing on optimal reconstruction have been propozed [130, 153], and improved performance has been obtained by also optimizing for discrimination [103, 102]. More recently [69] suggested to use sparse dictionaries together with an user-initiated active contour.

Our approach is closely related to the methods in [46] and [47]. These employ a dictionary that encodes patch-based self-similarities in the image for evolving a deformable boundary. In [46] a snake curve of is used. Consequently, only a single closed curve of a constant resolution may be tracked. These issues are alleviated in [47], where a level set is employed. However, multi-phase forces

used there are based on heuristics, and implicit curve representation lacks the compactness and gives only limited possibilities for incorporating shape priors.

7.2 Method

Our method fits in the framework of deformable models, where a curve is evolved in an image to obtain the segmentation. In our current implementation the initialization is provided as a curve (usually a small circle) vaguely outlining a number of regions in the image. This is an initial partitioning (we also call it labeling) of the image into K regions, where one region usually represents a background. Using a patch-based self-similarity model, any partitioning of the image can be transformed into pixel-wise probabilities of belonging to each of the K regions. Those probabilities are then used to evolve the outline. By sequentially updating the probabilities and the outline, we segment the image.

Two rather independent elements are vital for our approach. The first concerns the representation of the segmentation boundaries and the segments. For this we utilize the deformable simplical complex (DSC) framework [110], which provides an explicit curve with the topological adaptivity. In the context of image segmentation, DSC has only been used with intensity-based forces. In this work we combine DSC with the more general, probabilistic forces.

The second key element of our method is the model for encoding self-similarity in the image, which we use for computing curve deformation forces. A related model has been used for evolving active contours [46, 47]. In this work we provide a more efficient clustering algorithm, efficient probability update, and the derivation of forces on explicit curve.

In the upcoming description of our method, we start with the DSC based representation for general segmentation forces. Then we derive the segmentation forces used in our method.

7.2.1 Deformable adaptive mesh for image segmentation

DSC [6] is a generic method for interface tracking with applications in fluid simulation [113] and topology optimization [37]. When provided with the forces on the interface, DSC will handle the topology change automatically, including region merging and splitting. In this section we describe the DSC based curve representation, and how it can be used for image segmentation.

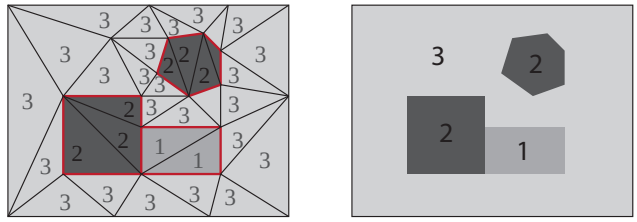


Figure 7.1: Mesh based representation of segments and the segment boundaries. Edges constituting the boundary are shown in red. Note the segment containing two disjoint regions and the two junction vertices.

Here we assume that the deforming forces are somehow computed from the image, but we are not interested in the nature of those forces. Later, in the next section, we will define the deformation forces used for the segmentation method proposed in this paper.

7.2.1.1 Mesh based representation of segments and boundaries

The DSC based representation relies on an irregular triangle mesh covering the image. Each triangle in the mesh is given a label from 1 to K that indicates to which segment it belongs. The segment boundaries in this representation are simply the edges shared by triangles that have different labels. Edges and vertices defining the boundary are called interface edges and interface vertices, see Fig. 7.1.

This representation provides an explicit curve, defined by the interface vertex positions $\mathbf{v}_i$. However, unlike simple snakes, the region information is encoded in triangle labels, preventing the problems with self-intersections and allowing for topology change. Furthermore, the representation naturally support arbitrary number of segments.

7.2.1.2 Evolving the segmentation boundary

Mesh-based segmentation boundary is explicit, defined by the interface vertex positions $\mathbf{v}_i$. The boundary deforms under influence of external and internal forces

$$\frac{\partial \mathbf{v}_i}{\partial t} = \mathbf{F}_{\text{ext}}(\mathbf{v}_i) + \mathbf{F}_{\text{int}}(\mathbf{v}_i)\,. \tag{7.1}$$

The internal forces are computed from the curve, typically discourage stretching (elasticity term) and bending (rigidity term) of the curve. We use only the elasticity regularization term, with internal forces defined as

$$\mathbf{F}_{\text{int}}(\mathbf{v}_i) = \sum_{e_{ij} \in \mathcal{N}_i} \frac{\mathbf{v}_j - \mathbf{v}_i}{\|\mathbf{v}_j - \mathbf{v}_i\|}, \tag{7.2}$$

where summation runs over $\mathcal{N}_i$ that contain interface edges e_{ij} adjacent to vertex i.

The external force is computed from the image and should pull the curve towards the desired segmentation. For now, let us assume that for every point $\mathbf{x}$ along the boundary we have the means of computing the magnitude of the external force in direction of outward pointing normal, denoted $f(\mathbf{x})$. The force on the interface vertex is then computed by integrating and distributing f according to

$$\mathbf{F}_{\text{ext}}(\mathbf{v_i}) = \sum_{e_{ij} \in \mathcal{N}_i} \mathbf{n}_{ij} \int_0^1 f(\mathbf{v}_i + s(\mathbf{v}_j - \mathbf{v}_i))(1-s)\mathrm{d}s, \tag{7.3}$$

where $\mathbf{n}_{ij}$ is an outward pointing normal to the edge e_{ij}.

To evolve the segmentation boundary, we apply the small displacement to the boundary vertices according to the forces

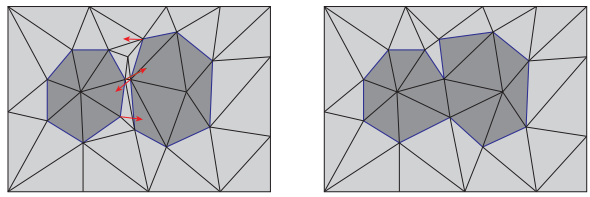
$$\mathbf{v}_i^{t+\Delta t} = \mathbf{v}_i^t + \mathbf{F}(\mathbf{v}_i^t)\Delta t\,. \tag{7.4}$$

When provided with the displacements of the interface vertices, DSC will move the interface and handle the topology change automatically. If, for example, the two interfaces components collide this change will cause them to merge, see Fig. 7.2.

The key to this topological adaptivity lies in a series of mesh updates performed with each deformation of the interface. The interface points are not moved to their destination in a single step. Instead, each points is moved in the direction of the destination as far as possible without inverting triangles. This is followed by the mesh improvement routine, and then moving points further until they eventually reach their destination. The reader may refer to [110] for more details.

7.2.1.3 Relabeling triangles

Curve evolution given by the internal and external forces may trigger a merge, a split, or a disappearance of a region, but not an insertion. To allow for an



(a) User defined velocity **(b)** Deformed mesh

Figure 7.2: Topology change in DSC during deformation: regions are merged when their interfaces collide. The red arrows show the displacements of the vertices.

insertion of the region, we may want to supplement the curve evolution with the event of a triangle changing a label. Turning this feature on and off gives us the control of the segmentation. For example, with relabeling turned off and the curve initialized inside the region to be segmented, the segmented region will grow but stay connected.

If relabeling is desired, we need to define a distance measure between an image pixel and a segmentation label. When integrated over the triangle, this gives the label of the segmentation region that the triangle should be assigned to.

7.2.1.4 Adapting the mesh resolution

A big advantage of our explicit scheme is that we can define a number of discrete events for adapting the triangle sizes as needed. These splitting and merging events are data-driven and triggered when some quality of the mesh entities (edges or faces) falls below the pre-set threshold.

Adaptivity should only resolve the situations which can not be resolved by moving interface vertices. Therefore we adapt only those entities which are not moving.

An interface edge is split if the line integral of external forces f from (Eq. 7.3) is larger than a energy threshold. Two subsequent interface edges are merged when curvature is below the threshold.

Splitting of the triangles is closely related to triangle relabeling. If the distance to the segmentation label, which the triangle is assigned to, is not low over the whole triangle, a split is performed.

After subdividing the mesh and capturing small features, we may have redundant vertices. To clean up these vertices and mantain the quality of the mesh we perform mesh thinning.

7.2.2 Self-similarity model

Our aim is to obtain a segmentation where similar patterns in the image belong to the same segment. To obtain this we encode self-similarity in the image by extracting patches of size $M \times M$ from the image, and then grouping similar image patches using clustering. The idea is that the patches that group together should have a similar segmentation. However, since patches are overlapping, every pixel is influenced by multiple clusters. Likewise, every cluster is influenced by all its patches.

In our approach we utilize the relation between the image and the clusters to transform a given image labeling into pixelwise label probabilities, similarly to [47]. We improve upon [47] by utilizing a variant of k-means trees for clustering, which results in a more efficient implementation of the algorithm.

7.2.2.1 Clustering the image patches

Intensity-based clustering can result in that image patches are not grouped together even though they contain similar patterns. They can however still obtain the same labeling as long as the clusters that they belong to have similar label probabilities. Our experience is therefore that it is an advantage to have very large number of clusters, while the precision in clustering is less important. Therefore, we have chosen to use a k-means tree [123]. A k-means tree is a graph build from consecutive k-means clusterings resulting in a directed rooted tree with a fixed branching factor b and number of layers t.

In order to limit the computational burden and memory usage when building the k-means tree we extract a subset of patches of size $M \times M$ from the image and collect pixel intensities in vectors of length $\rho = M^2 l$ (l is the number of channels in the image, e.g. $l = 1$ for grayscale images and $l = 3$ for RGB images). These are clustered into b clusters and each cluster center makes up a node in the tree. k-means clustering is repeated for all image vectors clustered to a node resulting in b new child nodes. This is repeated for all nodes until the desired number of layers, t, is reached. If a node contains less image vectors than the branching factor b, then no further clustering is carried out and child

nodes are marked as empty. In total n tree nodes indexed by $k \in \{1, \dots n\}$ are obtained.

Each k-means clustering is initialized by choosing a random subset of b image vectors and clustering is obtained by iteratively updating these centers. Our experience is that good performance is obtained without running the k-means until convergence, and therefore a fixed number of iterations is chosen, e.g. 10 iterations.

The outcome of the clustering is an assignment image, which for every pixel indicates to which cluster the patch centered around the pixel belongs to.

7.2.2.2 Transforming labeling to probability

For any given labeling of the image pixels, e.g. labeling given by the initialization, we can compute the pixelwise probabilities of belonging to the label. The detailed description of this approach can be found in [47].

The computation is based on two steps. Since we know the labeling of all patches in the image, in the first step we compute the pixelwise label probability for a cluster from the occurrence of a given label in each pixel in the cluster. In the second step we use the label probabilities in the cluster and go back to the image and computing the label probabilities in the image by averaging overlapping image patches.

7.2.2.3 Probability-based deformation forces

We can utilize the probability image to evolve the mesh-based segmentation representation. For each point $\mathbf{x}$ on the interface between the two segments k and k' we look up the pixel-wise probabilities of belonging to those segments. The deformation force $f(\mathbf{x})$ is then defined as

$$f(\mathbf{x}) = P_k(\mathbf{x}) - P_{k'}(\mathbf{x})\,. \tag{7.5}$$

Discrete events are also defined in terms of pixel-wise probabilities. The energy measure of the triangle being labeled as belonging to segment k is obtained by integrating P_k along the triangle. In case of a low probability, the triangle will be relabeled. In case of a high variance, the triangle will be subdivided.

7.3 Results and discussion

We have carried out experiments on natural images and composed textures in order to illustrate the properties of our method. A number of examples is shown in Fig. 7.3, illustrates how the method allows for multiple labels with 6 and 5 labels respectively in the images in Fig. 7.3 (a-d). In Fig. 7.3 (e-h) we demonstrate that allowing for relabeling has effect if multiple disjoint regions of similar texture are present in the image. In Fig. 7.4 we show segmentation obtained on natural images with the mesh overlayed. This illustrates how the mesh adapts to details in the images, such that small triangles are present at smaller details, like in the tails of the cats, whereas in larger regions the obtained triangles are much larger.

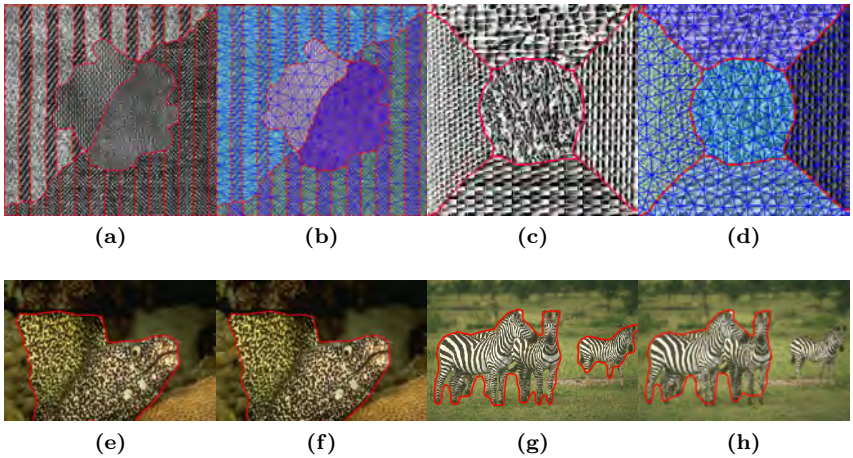
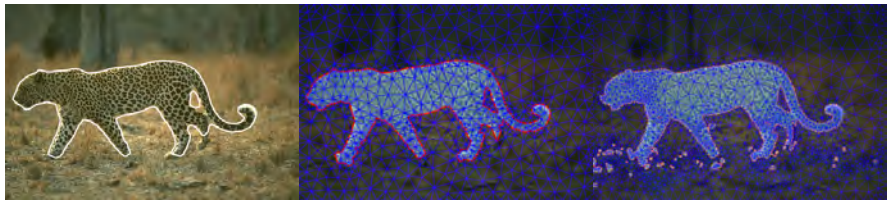
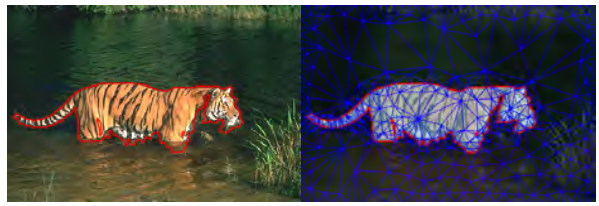


Figure 7.3: Segmentation examples showing the obtained segmentations on different images. (a-d) are synthetically composed images where the algorithm was run allowing for triangle relabelling. (e-h) shows results obtained on natural images showing the difference between triangle relabelling (e,g) and no relabeleling (f,h). Note how the segmentation of the two groups of zebras in (g) where relabelling is allowed, whereas only the one group is segmented if (f) when no relabelling is allowed.

In Fig. 5.15 we show a small comparison between the proposed method and the method in [46], [47], and [69]. In this experiment we have used the same implementation of the patch based self-similarity model, as the one used for the proposed method, for [46] and [47]. In [46] the boundary is represented using a snakes model that does not allow for topological adaptivity, and in [47] a level



(a) (b) (c)



(d) (e)

Figure 7.4: Segmentation examples showing the boundary and the mesh overlay. (a,d) and (b,e) are without triangle relabelling whereas (c) is with. Note how the mesh resolution adapts to the details of the image.

sets is used with an implicit boundary allowing for topological adaptivity. The model in [69] trains a texture representation using a two class sparse dictionary. All methods obtain similar results, but it is important to note the limited input in the models employing the patch-based self-similarity model, whereas a substantial amount of information is given in the initializations from [69].

If disjoint objects are to be segmented, like the image of lions or zebras, it is advantageous to use topological adaptivity, because it allows the objects of the same type to be segmented to have the same label. However, topological adaptivity also gives more flexibility to the model, which in some cases can be a disadvantage, as seen in the bottom image in Fig. 7.5, where the grey bottom part of the images is labelled the same as the kangaroo. This is both seen for the proposed method and the level set based model [47].

All segmentations using the patch based self-similarity model have been run with the same set of parameters for obtaining the tree based clustering. The patch size was chosen to be 5×5, and the tree was 4 layers with a branching factor of 7. 20000 patches were randomly chosen for building the clustering tree. All methods used the same smoothing factor (the parameter was set to $\alpha = 2$ in the proposed model as well as in [46] ($\beta = 2$) and [47] (smoothing factor $= 2.$)). Runtimes were approximately 20 seconds for the proposed method, 40

seconds for [46] and 25 seconds for [47]. Both [46] and [47] are implemented in Matlab using mex-files implemented in C++ whereas the proposed method is only C++.

7.4 Conclusion

In this paper we have proposed a segmentation method that utilizes a deformable triangle mesh for image segmentation. This allows for the model flexibility advantages of an explicit mesh while retaining the ability of topological adaptivity. In this paper we have provided a detailed description of the method and demonstrated some of its properties through example experiments that shows a similar performance to using an explicit snake boundary model without topological adaptivity or topology adaptive level sets. But with the proposed method it can be chosen if one or multiple objects should be segmented. Our plan is to extend this work to volumetric segmentation and employ shape priors for the segmentation.

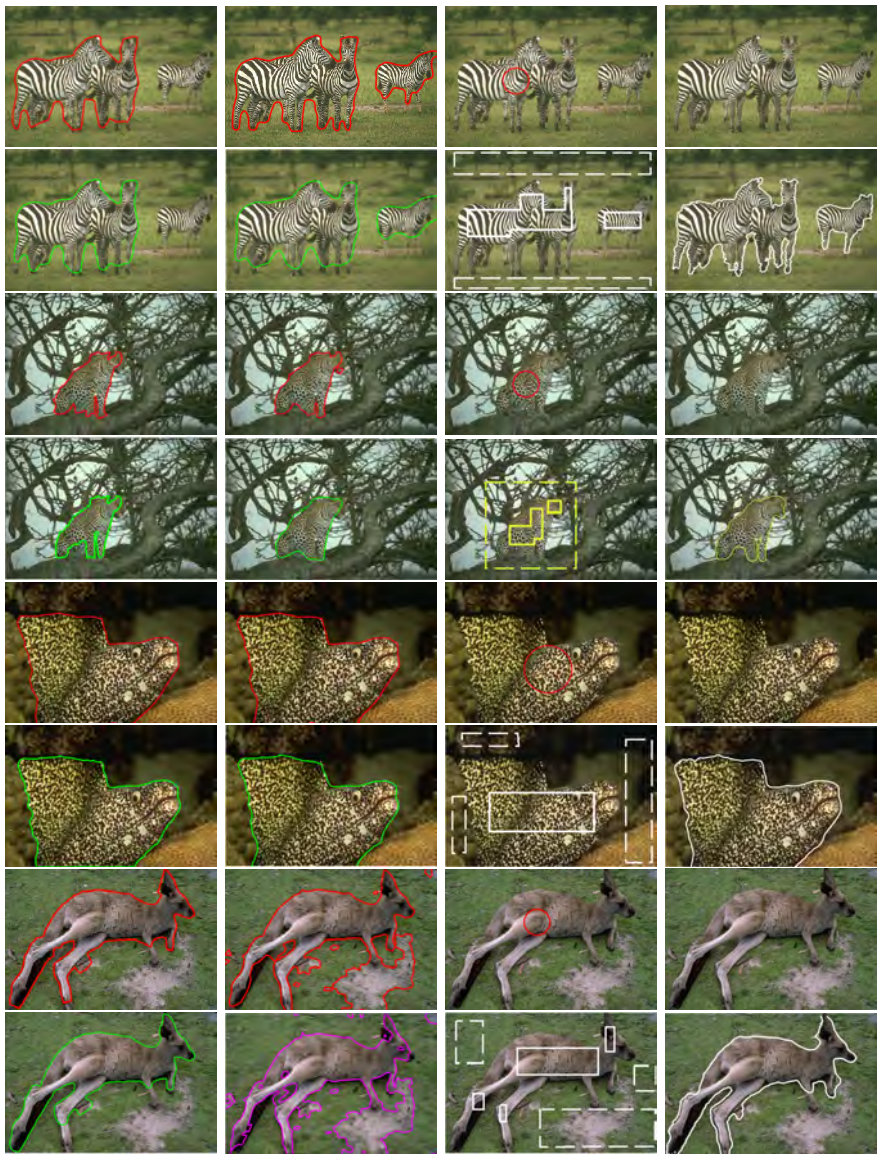


Figure 7.5: Segmentation comparison between different methods. There are two rows for each images, where the top row from left shows the proposed method without triangle relabelling, proposed method with triangle relabelling, and initialization (used for the four images from the left), and original. Bottom row shows result by [46], result by [47], initialization and result by [69].

Chapter 8

Fluid Tracking with the DSC

This chapter describes our work in fluid tracking using the DSC. We follow [179], a method for one-phase fluid tracking, and modify the algorithm to make it work with multi-phase fluid tracking. We perform two experiments: the dam break problem and a simulation of two-phase fluid.

8.1 Particle approximation

In fluid particle, the fluid is discretized to a set of particles $\mathcal{F} = \{b\}$, each carries physical information, e.g. pressure p, density ρ, and mass m. Value not at a particle position will be approximated using particle approximation.

$$f(\mathbf{r}) = \sum_{b \in \mathcal{F}} \frac{m_b}{\rho_b} f_b W_b(\mathbf{r}) \tag{8.1}$$

where $f(\mathbf{r})$ is the approximating value at position $\mathbf{r}$, f_b is value carried by partible b, and $W_b(\mathbf{r})$ is the kernel function. In this thesis, we utilize Wendland kernal

$$W_b(\mathbf{r}) = \begin{cases} \frac{21}{16\pi h^3}(1 - \frac{q}{2})^4(1 + 2q) & 0 \leq q \leq 2 \\ 0 & 2 < q \end{cases} \tag{8.2}$$

where h is the influence radius of a particle, and $q = \frac{\|\mathbf{r}-\mathbf{r}_b\|}{h}$.

8.2 Particles surface defined by an anisotropic kernel

The fluid surface can be determine as an iso-surface of a scalar field [117]

$$\phi(\mathbf{r}) = \sum_{b\in\mathcal{F}} \frac{m_b}{\rho_b} W_b(\mathbf{r}) \tag{8.3}$$

In order to improve the quality, we follow [178] and replace the kernel W_b with an anisotropic kernel by replacing h with a 3×3 matrix $\mathbf{G}$

$$W'_b(\mathbf{r}) = \frac{21}{16\pi} det(\mathbf{G}_b) \begin{cases} (1-\frac{q'}{2})^4(1+2q') & 0 \le q' \le 2 \\ 0 & 2 < q' \end{cases} \tag{8.4}$$

where $q' = \big\|\mathbf{G}_b(\mathbf{r}-\mathbf{r}_b)\big\|$.

The matrix $\mathbf{G}_b$ is determined by weighted principal component analysis to the neighbor particles $\mathcal{N}$ of the particle b. To compute $\mathbf{G}_b$, first we compute a weighted covariance matrix of the neighbor particles in $\mathcal{N}$

$$\mathbf{C}_b = \sum_{b_i\in\mathcal{N}} \omega_{bb_i}(\mathbf{r}_{b_i} - \overline{\mathbf{r}}_b)(\mathbf{r}_{b_i} - \overline{\mathbf{r}}_b)^T / \sum_{b_i\in\mathcal{N}} \omega_{bb_i} \tag{8.5}$$

where $\overline{\mathbf{r}}_b$ is the weighted average position

$$\overline{\mathbf{r}}_b = \sum_{b_i\in\mathcal{N}} \omega_{bb_i}\mathbf{r}_{b_i} / \sum_{b_i\in\mathcal{N}} \omega_{bb_i} \tag{8.6}$$

ω_{bb_i} is an isotropic weighting function between particle b and b_i with support h_s

$$\omega_{bb_i} = \begin{cases} 1 - \left(\frac{\|\mathbf{r}_b - \mathbf{r}_{b_i}\|}{h_s}\right)^3 & \|\mathbf{r}_b - \mathbf{r}_{b_i}\| < h_s \\ 0 & \text{otherwise} \end{cases} \tag{8.7}$$

Using singular value decomposition of matrix $\mathbf{C}_b$ yields

$$\mathbf{C}_b = \mathbf{R}\mathbf{\Sigma}\mathbf{R}^T \tag{8.8}$$

where $\mathbf{R}$ represents the rotation, and $\mathbf{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \sigma_3)$ represents scaling factors. $\mathbf{C}_b$ is then scaled by

$$\widetilde{\mathbf{C}}_b = \mathbf{R}\widetilde{\Sigma}\mathbf{R}^T \tag{8.9}$$

where $\widetilde{\Sigma}$ is

$$\widetilde{\Sigma} = \begin{cases} k_s \text{diag}(\widetilde{\sigma}_1, \widetilde{\sigma}_2, \widetilde{\sigma}_3) & \text{size}(\mathcal{N}) > N_\epsilon \\ k_n \mathbf{I} & \text{otherwise} \end{cases} \tag{8.10}$$

here $\widetilde{\sigma}_i = \max(\sigma_k, \sigma_1/k_r)$. k_s, k_n, k_r are constant parameters. Choices of these parameters depend on the problem.

Finally we achieve the matrix $\mathbf{G}_b$

$$\mathbf{G}_b = \frac{1}{h}\mathbf{R}\widetilde{\Sigma}^{-1}\mathbf{R}^T \tag{8.11}$$

8.3 Surface tracking with the DSC

Algorithm 15: Fluid tracking with the DSC

```
1  while simulation is on do
2  |
3  |  Run GPUSPH
4  |
   |  /* Step 1: Advection                                   */
5  |  Add ghost particles
6  |  foreach interface vertex v do
7  |  |  Interpolate velocity of v with particle approximation
8  |  Deform the mesh with the DSC
9  |
   |  /* Step 2: Projection                                  */
10 |  Compute anisotropic field φ
11 |  foreach interface vertex v do
12 |  |  Project v to the isosurface of φ
13 |  Deform the mesh with the DSC
```

We utilize GPUSPH [12, 1], an open source framework for smooth particle hydrodynamics using GPU, and follow [179] for explicit interface tracking using anisotropic kernel. The algorithm proposed in [179] is originally used for one-phase fluid simulation. We apply some modifications to enable the method for multi-phase tracking. Finally, we utilize Misuba renderer [79] for transparency rendering.

The tracking algorithm includes two steps (Alg. 15). First, we advect the interfaces following the movement of the particles by interpolating velocities of interface vertices using particle approximation in Eq. 8.1. Be note that the mesh interfaces are the boundary of the fluid, and boundary treatment is required. In our algorithm, we employ the ghost particle technique that creates unreal particles reflecting boundary particles. The second step projects the mesh interfaces to the anisotropic surfaces of the particles to reduce cumulative error. This step does not need to run in every iteration. Normally we perform projection once the interfaces have moved by a distance equal to the influence radius h.

Algorithm 16: Projection to the isosurface of ϕ

Input: vertex position $\mathbf{p}$, normal $\mathbf{n}$, search radius r, ϕ

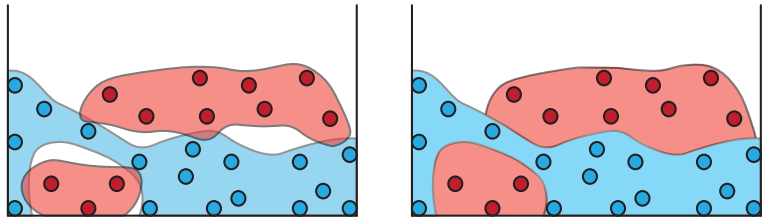
Search point $\mathbf{p}_2$
if $\mathbf{p}$ *is inside* **then**
  $\mathbf{p}_2 = \mathbf{p}_1 + r\mathbf{n}$
else
  $\mathbf{p}_2 = \mathbf{p}_1 - r\mathbf{n}$

if $\mathbf{p}$ *and* $\mathbf{p}_2$ *are on same side* **then**
  `/* Fail to project.  Move the vertex on the normal direction  */`
  Return $\mathbf{p}_2$

for *Five itterations* **do**
  $\mathbf{p}_{\text{center}} = \frac{1}{2}(\mathbf{p} + \mathbf{p}_2)$
  if $\mathbf{p}_{\text{center}}$ *and* $\mathbf{p}$ *are on same size* **then**
    $\mathbf{p} = \mathbf{p}_{\text{center}}$
  else
    $\mathbf{p}_2 = \mathbf{p}_{\text{center}}$
Return $\frac{1}{2}(\mathbf{p} + \mathbf{p}_2)$

The scalar field $\phi(\mathbf{r})$ provided by the anisotropic can only determines if the point $\mathbf{r}$ is outside ($\phi(\mathbf{r}) = 0$) or inside ($\phi(\mathbf{r}) > 0$). To compute the projection of a interface vertex to the isosurface, we follow [33] that performs binary search on the normal direction (Alg. 16).

The anisotropic kernel can provide tight surfaces of the particle. However, in case of multi-phase fluid, there may be vacuum and overlapping between phases. We apply phase priority for projection (step 2), *i.e.* if the interface vertex is close to multiple fluids, we project it to the phase with higher priority.



(a) Isosurface of individual fluid **(b)** Prior to blue fluid

Figure 8.1: Anisotropic surface of multiple fluids

8.4 Dam break simulation

Tab. 8.2 shows the parameters for SPH simulation. Visual rendering of the fluid is shown in Fig. 8.2. Performance of the experiment is shown in Tab. 8.1.

Table 8.1: Simulation information

# particles	12.8k	# iterations	995
Meshing time	21 seconds	Anisotropic kernel	0.6 second
Max # of DSC iteration	5		

Table 8.2: Simulation parameters for dam break problem. k_s, k_n, k_r are the parameters for anisotropic kernel

Domain size	$1.6 \times 0.67 \times 0.6$	Fluid init size	$0.4 \times 0.67 \times 0.4$
Particle spacing	0.02	Particle radius h	0.026
Influence radius	0.052	Fluid density	1000
k_s, k_n, k_r	3000, 0.35, 4	Simulation time	1.5 seconds

Table 8.3: Average mesh statistic

Number of					
nodes	edges	faces	tetrahedra	interface nodes	interface faces
87k	570k	953k	470k	17.3k	34.8k

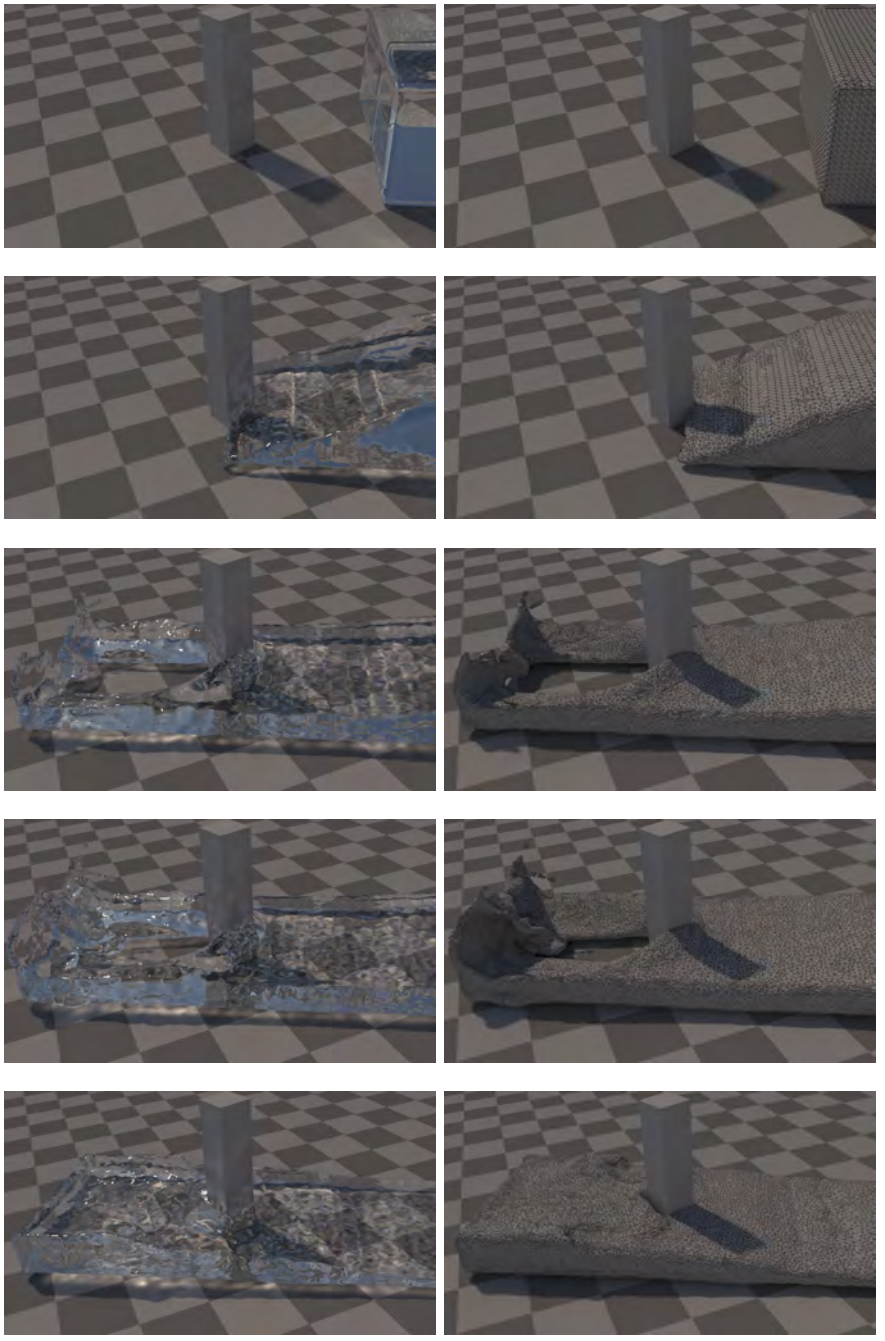


Figure 8.2: Dam break simulation

8.5 Two-phase fluid simulation

Tab. 8.4 shows the parameters for SPH simulation.

Table 8.4: Simulation parameters for two-phase fluid problem. k_s, k_n, k_r are the parameters for anisotropic kernel

Domain size	$0.15 \times 0.15 \times 0.15$	Fluid init size	$0.15 \times 0.15 \times 0.054/0.125$
Particle spacing	0.002	Particle radius h	0.0032
Influence radius	0.0064	Fluid density	1000/200
k_s, k_n, k_r	160000, 0.35, 4	Simulation time	3 seconds

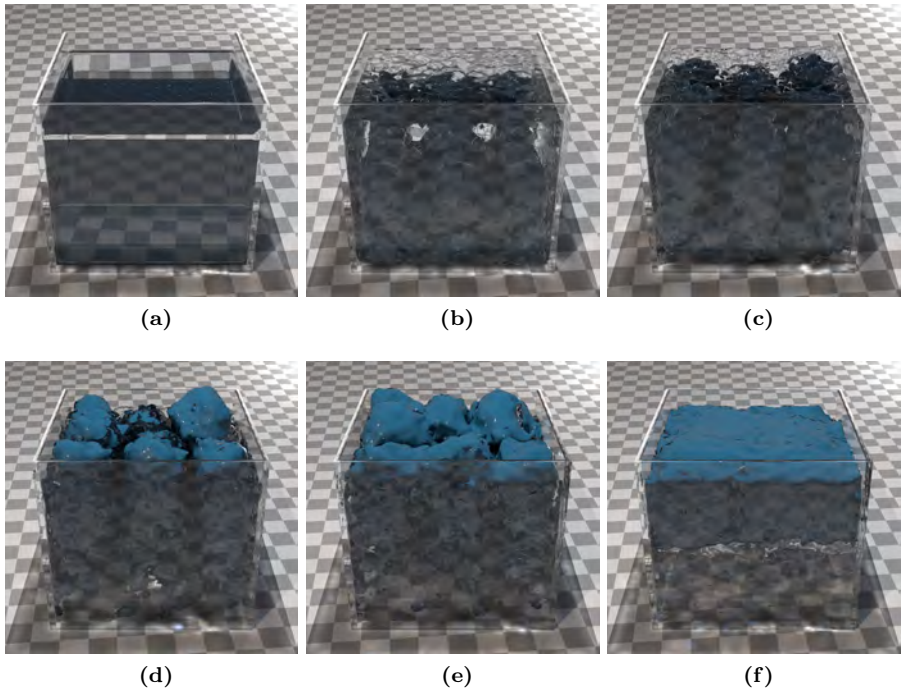


Figure 8.3: Fluid rendering. Phase 1 (lighter) is the green fluid. Phase 2 (heavier) is the translucent water.

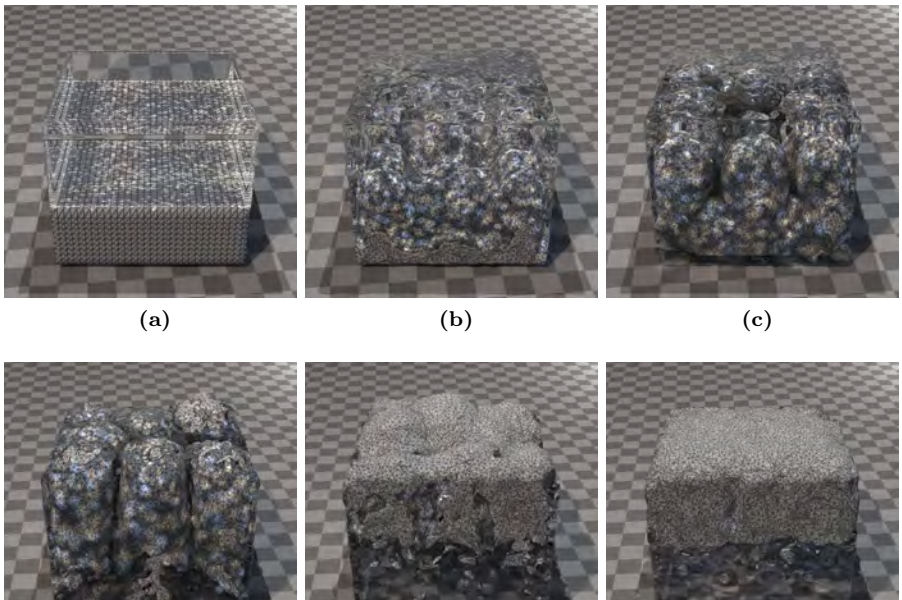
Fig. 8.3 shows the rendering of the fluid, and Fig. 8.4 shows the mesh frame of phase 1. The computation time of the experiment is shown in Tab. 8.6.

Table 8.5: Average mesh statistic

Number of					
nodes	edges	faces	tetrahedra	interface nodes	interface faces
224k	1.5M	2.5M	1.3M	70k	140k

Table 8.6: Simulation information

# particles	180k	# iterations	1800
Meshing time	160 seconds	Anisotropic kernel	5 seconds
Max # of DSC iteration	5		



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 8.4: Fluid rendering with phase 1 is the mesh frame

8.6 Discussion

We follow the two-phase fluid tracking method proposed by Yu *et al.* [179] with some modifications to make the algorithm work with multi-phase fluid. By utilizing the anisotropic kernel, we can track the surface accurately (see Fig. 8.5).

Figure 8.5: Fluid surface and particles. Iteration 600 / 900

The two experiments of dam break and two-phase fluid demonstrate the capability of the DSC as a multi-phase fluid tracking method that can handles complex topology. On the other hand, the DSC is the bottleneck of the performance. This is due to the simplicity of particle-based method and the trade off for the high accuracy and multi-phase support of the DSC.

Compare to the previous fluid simulation with the DSC [114], tracking particle fluid is more promising in term of performance. In our experiment, the computation time of SPH is significantly smaller than the DSC advection time, whereas in the FEM fluid [114], solving the fluid dynamics consumes around 70% of total computation time.

Chapter 9

Conclusion and outlook

The Deformable Simplicial Complex (DSC) is an explicit interface tracking method, which was introduced to promote the usages of Lagrangian representations (where the interfaces are defined explicitly, and interface nodes move with the materials) for solving problems. This thesis utilizes the DSC to solve image segmentation and fluid tracking problems. Besides, we propose a method for DSC performance optimization.

The first part of the thesis is a novel method for 2D/3D image segmentation, which is combined with mesh generation. We have shown that our method possesses several advantages, including: multi-phase support, due to the nature of triangle/tetrahedral mesh; higher accuracy than the level set method, as we distinguish image space (fixed grid) and segmentation space (meshes); comparable or faster than the level set method in term of performance; and adaptive output meshes, which are valuable for analysis and simulation. An implementation of 3D intensity-based segmentation in `C++` is available to publicity at `https://github.com/tuannt8/3D-image-segmentation` [121].

Having that said, the proposed method still has limitations: open curves/surfaces is not supported and low performance. Of these two, performance is at higher concern, especially in 3D. For further development, we would like to have more investigation on a couple of things. First, we would like to extend the code in [121] for various input modalities beside intensity. Second, we want to promote

the method to public, so it can be utilized in various practical applications.

The second part of the thesis is a method for performance optimization for iterative-meshing frameworks such as the DSC. We propose a caching scheme that allows us to balance the memory usage and the performance dynamically based on the problems' requirements. Our experiments show that this caching scheme can reduce computation time up to 80% with up to 50% memory overhead. Furthermore, caching entity attributes helps enabling parallel meshing using color method. The shortcomings of the method is that choosing the topological relations for caching requires profiling of frequently queried data. Besides, the caching relations are limited to the link of the key entity.

The last part of the thesis is our work on fluid tracking using the DSC. Our experiments have shown the capability of the DSC for multi-phase fluid tracking with complex topology. In comparison to the previous fluid simulation using the DSC, our approach can be four times faster in term of performance of the whole algorithm (fluid dynamics + advection). For further development, it would be interesting to see how the DSC compared to other interface tracking methods, especially those that support multi-phase.

In conclusion, we utilize the DSC not because the framework is available to us but due to its unique properties in multi-phase support and interior mesh representation. As shown in the image segmentation method, examination of interior regions is crucial in order to fully minimize the global Mumford-Shah energy functional. Besides, tetrahedral mesh is highly attractive for analysis and simulation.

Bibliography

[1] GPUSPH.

[2] Tyler J. Alumbaugh and Xiangmin Jiao. Compact Array-Based Mesh Data Structures. In *Proceedings of the 14th International Meshing Roundtable*, pages 485–503. Springer Berlin Heidelberg, 2005.

[3] L. Ambrosio. *A Compactness Theorem for a Special Class of Functions of Bounded Variation*. Preprints di matematica. Scuola normale superiore, 1988.

[4] Luigi Ambrosio and Vincenzo Maria Tortorelli. Approximation of functional depending on jumps by elliptic functional via t-convergence. *Communications on Pure and Applied Mathematics*, 43(8):999–1036, 1990.

[5] Authors. Multi-phase image segmentation with the adaptive deformable mesh. *Submitted to IEEE Transactions on Image Processing, provided in suplementary material*, 2016.

[6] J. Andreas Bærentzen. Deformable Simplicial Complex. *https://github.com/janba/2D-DSC*.

[7] J. Andreas Bærentzen. Deformable Simplicial Complex (DSC) method, 2013.

[8] Leah Bar, Tony F. Chan, Ginmo Chung, Miyoun Jung, Nahum Kiryati, Rami Mohieddine, Nir Sochen, and Luminita A. Vese. Mumford and Shah Model and its applications to image segmentation and image restoration. In *Handbook of Mathematical Methods in Imaging*, pages 1097–1154. 2011.

[9] Adam W. Bargteil, Tolga G. Goktekin, James F. O'brien, and John a. Strain. A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1):19–38, 2006.

[10] Bruce G Baumgart. Winged edge polyhedron representation. Technical report, DTIC Document, 1972.

[11] Mark W Beall and Mark S Shephard. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering*, 40(9):1573–1596, 1997.

[12] Giuseppe Bilotta, Alexis Herault, Annalisa Cappello, Gaetana Ganci, and Ciro Del Negro. GPUSPH: a Smoothed Particle Hydrodynamics model for the thermal and rheological evolution of lava flows. *Geological Society, London, Special Publications*, 426:387–408, 2014.

[13] Andrew Blake and Andrew Zisserman. *Visual reconstruction*. MIT press, 1987.

[14] Blaise Bourdin. Image segmentation with a finite element method. *Mathematical Modelling And Numerical Analysis*, 33:229–244, 1999.

[15] Jörg Bredno, Thomas M. Lehmann, and Klaus Spitzer. A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):550–563, 2003.

[16] Claude R Brice, Claude L Fennema, and C Fenema. Scene analysis using regions. *Artificial intelligence*, 1(3-4):205–226, 1970.

[17] Robert Bridson. *Fluid Simulation*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[18] Tyson Brochu and Robert Bridson. Robust Topological Operations for Dynamic Explicit Surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[19] Thomas Brox, Mikaël Rousson, Rachid Deriche, and Joachim Weickert. Colour, texture, and motion in level set based segmentation and tracking. *Image and Vision Computing*, 28(3):376–390, 2010.

[20] Thomas Brox and Joachim Weickert. A tv flow based local scale measure for texture discrimination. In *Computer Vision-ECCV 2004*, pages 578–590. Springer, 2004.

[21] Thomas Brox and Joachim Weickert. A tv flow based local scale estimate and its application to texture discrimination. *Journal of Visual Communication and Image Representation*, 17(5):1053–1073, 2006.

[22] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges-A scalable representation for triangle meshes. *Journal of Graphics tools*, 3(4):1–11, 1998.

[23] Marcel Campen and Leif Kobbelt. Exact and robust (self-)intersections for polygonal meshes. *Computer Graphics Forum*, 29(2):397–406, 2010.

[24] Viccnt Caselles, Francine Catt6, Tomeu Coil, and Fran˜oise Dibos. A geometric model for active contours in image processing. *Mathematics Subject Classification*, 66:1–31, 1993.

[25] Vicent Caselles, Francine Catté, Tomeu Coll, and Françoise Dibos. A geometric model for active contours in image processing. *Numerische mathematik*, 66(1):1–31, 1993.

[26] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *IJCV*, 22(1):61–79, 1997.

[27] Waldemar Celes, Glaucio H. Paulino, and Rodrigo Espinha. A compact adjacency-based topological data structure for finite element mesh representation. *International Journal for Numerical Methods in Engineering*, 64(11):1529–1556, 2005.

[28] Antonin Chambolle. Finite-differences discretizations of the Mumford-Shah functional. *Mathematical Modelling and Numerical Analysis*, 33(2):261–288, 1999.

[29] Antonin Chambolle and Gianni Dal Maso. Discrete approximation of the Mumford-Shah functional in dimension two. *Mathematical Modelling and Numerical Analysis*, 33(4):651–672, 1999.

[30] Tony Chan and Luminita Vese. An active contour model without edges. In *Scale-Space Theories in Computer Vision*, pages 141–151. Springer, 1999.

[31] Tony F. Chan and Luminita A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277, 2001.

[32] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

[33] Nuttapong Chentanez, Matthias Müller, Miles Macklin, and Tae-yong Kim. Fast grid-free surface tracking. *ACM Transactions on Graphics*, 34(4):148:1–148:11, jul 2015.

[34] Andrey N Chernikov and Nikos P Chrisochoides. Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity. *SIAM Journal on Scientific Computing*, 33(6):3491–3508, 2011.

[35] Jatin Chhugani and Subodh Kumar. Geometry engine optimization: cache friendly compressed representation of geometry. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 9–16. ACM, 2007.

[36] Asger Nyman Christiansen, J. Andreas Bærentzen, Morten Nobel-Jørgensen, Niels Aage, and Ole Sigmund. Combined shape and topology optimization of 3D structures. *Computers & Graphics*, 46:25–35, 2015.

[37] Asger Nyman Christiansen, Morten Nobel-Jørgensen, Niels Aage, Ole Sigmund, and Jakob Andreas Bærentzen. Topology optimization using an explicit interface representation. *Structural and Multidisciplinary Optimization*, pages 1–13, 2013.

[38] Laurent D Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 53(2):211–218, 1989.

[39] Laurent D. Cohen and Isaac Cohen. Finite-element methods for active contour models and balloons for 2-D and 3-D images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9212248):1131–1147, 1993.

[40] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[41] Stephanie Coleman and Kathryn S. McKinley. Tile size selection using cache organization and data layout. *ACM SIGPLAN Notices*, 30(6):279–290, 1995.

[42] D Louis Collins, Alex P Zijdenbos, Vasken Kollokian, John G Sled, Noor J Kabani, Colin J Holmes, and Alan C Evans. Design and construction of a realistic digital brain phantom. *IEEE transactions on medical imaging*, 17(3):463–468, 1998.

[43] Stuart D. Connell and D. G. Holmes. Three-dimensional unstructured adaptive multigrid scheme for the Euler equations. *AIAA Journal*, 32(8):1626–1632, 1994.

[44] G. R. COWPER. Gaussian quadrature formulas for triangles. *International Journal for Numerical Methods in Engineering*, 7(3):405–408, 1972.

[45] Fang Da, Christopher Batty, and Eitan Grinspun. Multimaterial mesh-based surface tracking. *ACM Transactions on Graphics*, 33(4):1–11, jul 2014.

[46] Anders Bjorholm Dahl and Vedrana Andersen Dahl. Dictionary snakes. In *2014 22nd International Conference on Pattern Recognition (ICPR)*, pages 142–147. IEEE, 2014.

[47] Anders Bjorholm Dahl and Vedrana Andersen Dahl. Dictionary based image segmentation. In *Image Analysis, 19th Scandinavian Conference on Image Analysis (SCIA)*, pages 26–37. Springer, 2015.

[48] Vedrana Andersen Dahl, Asger Nyman Christiansen, and Jakob Andreas Bærentzen. Multiphase Image Segmentation Using the Deformable Simplicial Complex Method. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, number 2, 2014.

[49] Meizhong Dai and David P. Schmidt. Adaptive tetrahedral meshing in free-surface flow. *Journal of Computational Physics*, 208(1):228–252, 2005.

[50] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[51] Guillaume Damiand. Combinatorial Maps. In *{CGAL} User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016.

[52] Guillaume Damiand. Linear Cell Complex. In *{CGAL} User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016.

[53] Leila De Floriani, Annie Hui, Daniele Panozzo, and David Canino. A dimension-independent data structure for simplicial complexes. In *Proceedings of the 19th International Meshing Roundtable*, pages 403–420. Springer, 2010.

[54] H. Delingette. Simplex meshes: a general representation for 3D shape reconstruction. *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 856–859, 1994.

[55] Peter J Denning. Virtual Memory. *ACM Computing Surveys (CSUR)*, 2(3):153–189, 1970.

[56] Jian Du, Brian Fix, James Glimm, Xicheng Jia, Xiaolin Li, Yuanhua Li, and Lingling Wu. A simple package for front tracking. *Journal of Computational Physics*, 213(2):613–628, 2006.

[57] Ye Duan and Hong Qin. A subdivision-based deformable model for surface reconstruction of unknown topology. *Graphical Models*, 66(4):181–202, 2004.

[58] Alexandre Dufour, Nicole Vincent, and Auguste Genovesio. 3D Mumford-Shah based active mesh. *Progress in Pattern Recognition, Image Analysis and Applications*, 4225:208–217, 2006.

[59] Michael Elad. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer, 2010.

[60] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. *A Hybrid Particle Level Set Method for Improved Interface Capturing*, volume 183. 2002.

[61] Moffett Field, Moffett Field, Rupak Biswas, and Roger C Strawn. A new procedure for dynamic adaption of three-dimensional unstructured grids. *Applied Numerical Mathematics*, 13(6):437–452, 1994.

[62] M.A. Fischler and R.A. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, C-22(1):67–92, jan 1973.

[63] Leila De Floriani and Annie Hui. Data Structures for Simplicial Complexes: An Analysis And A Comparison. *Symposium on Geometry Processing*, 2005.

[64] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58(5):471–483, 1996.

[65] La Freitag and C Ollivier-Gooch. Tetrahedral Mesh improvement using swaping and smoothing. *International Journal for Numerical ...*, 40(November 1996):3979–4002, 1997.

[66] Lori Freitag, Mark Jones, and Paul Plassmann. An Efficient Parallel Algorithm for Mesh Smoothing. *INTERNATIONAL MESHING ROUNDTABLE*, pages 1–14, 1995.

[67] Lori Freitag, Mark Jones, and Paul Plassmann. A Parallel Algorithm for Mesh Smoothing. *SIAM Journal on Scientific Computing*, 20(6):2023–2040, jan 1999.

[68] Pascal Jean Frey and Paul L George. *Mesh generation: application to finite elements*. Wiley Online Library, 2008.

[69] Yi Gao, Sylvain Bouix, Martha Shenton, and Allen Tannenbaum. Sparse texture active contour. *TIP*, 2013.

[70] M.R. Garey and D.S. Johnson. *Computer and intractability*. W. H. Freeman, New York, 1979.

[71] Rao V. Garimella. Mesh data structure selection for mesh generation and FEA applications. *International Journal for Numerical Methods in Engineering*, 55(4):451–478, 2002.

[72] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[73] James Glimm, John W. Grove, Xiao Lin Li, Keh-ming Shyue, Yanni Zeng, and Qiang Zhang. Three-Dimensional Front Tracking. *SIAM Journal on Scientific Computing*, 19(3):703–727, 1998.

[74] James Glimm, John W Grove, Xiaolin L Li, and D C Tan. Robust Computational Algorithms for Dynamic Interface Tracking in Three Dimensions. *SIAM Journal on Scientific Computing*, 21(6):2240–2256, 2000.

[75] Rafael C Gonzalez and Richard E Woods. Digital image processing, 2012.

[76] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi. *ACM transactions on graphics (TOG)*, 4(2):74–123, 1985.

[77] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, 1985.

[78] Yasushi Ito, Alan M Shih, Anil K Erukala, Bharat K Soni, Andrey Chernikov, Nikos P Chrisochoides, and Kazuhiro Nakahashi. Parallel unstructured mesh generation by an advancing front method. *Mathematics and Computers in Simulation*, 75(5-6):200–209, 2007.

[79] Wenzel Jakob. Mitsuba renderer, 2010.

[80] M Jan and Leif Kobbelt. OpenFlipper : An Open Source Geometry Processing and Rendering Framework. In *International Conference on Curves and Surfaces*, pages 488–500, 2012.

[81] Mark T. Jones and Paul E. Plassmann. A Parallel Graph Coloring Heuristic. *SIAM Journal on Scientific Computing*, 14(3):654–669, may 1993.

[82] Y. KALLINDERIS and P. VIJAYAN. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA Journal*, 31(8):1440–1447, 1993.

[83] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes : Active Contour Models. *International Journal of Computer Vision*, 331(4):321–331, 1988.

[84] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, 1988.

[85] Bryan Matthew Klingner and Jonathan Richard Shewchuk. Aggressive tetrahedral mesh improvement. *Proceedings of the 16th International Meshing Roundtable, IMR 2007*, pages 3–23, 2008.

[86] Georges Koepfler, Christian Lopez, and Jean-Michel Morel. A multiscale algorithm for image segmentation by variational method. *SIAM journal on numerical analysis*, 31(1):282–299, 1994.

[87] M. Krause, J. M. Hausherr, B. Burgeth, C. Herrmann, and W. Krenkel. Determination of the fibre orientation in composites using the structure tensor and local X-ray transform. *Journal of Materials Science*, 45(4):888–896, 2010.

[88] M Kremer, D Bommes, and L Kobbelt. Open VolumeMesh–A Versatile Index-Based Data Structure for 3D Polytopal Complexes. *Proceedings of the 21st International Meshing Roundtable*, pages 531–548, 2013.

[89] J.-O. Lachaud and B. Taton. Deformable model with adaptive mesh and automated topology changes. *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, 2003.

[90] M Lage, T Lewiner, H Lopes, and L Velho. CHF: a scalable topological data structure for tetrahedral meshes. *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)*, pages 1–6, 2005.

[91] Rynson W.H. Lau, Rynson W.H. Lau, Oliver Chan, Oliver Chan, Mo Luk, Mo Luk, Frederick W.B. Li, and Frederick W.B. Li. A collision detection framework for deformable objects. *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '02*, page 113, 2002.

[92] S. C. Leemann, Å. Andersson, M. Eriksson, L.-J. Lindgren, E. Wallén, J. Bengtsson, and A. Streun. Beam dynamics and expected performance of Sweden's new storage-ring light source: MAX IV. *Physical Review Special Topics - Accelerators and Beams*, 12(12):120701, 2009.

[93] Tianhu Lei and Wilfred Sewchand. Statistical approach to X-ray CT imaging and its applications in image analysis. I. Statistical analysis of X-ray CT imaging. *IEEE transactions on medical imaging*, 11(1):53–61, 1992.

[94] Joshua Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. *VIS02 IEEE Visualization 2002*, pages 259–265, 2002.

[95] Stan Z Li. Markov random field models in computer vision. In *European conference on computer vision*, pages 361–370. Springer, 1994.

[96] Johan Lie, Marius Lysaker, and Xue-Cheng Tai. A variant of the level set method and applications to image segmentation. *Mathematics of Computation*, 75(255):1155–1175, 2006.

[97] M Luby. A simple parallel algorithm for the maximal independent set problem. *Annual ACM Symposium on Theory of Computing*, page 1, 1985.

[98] Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

[99] David MacDonald, David Avis, and Alan C. Evans. Multiple surface identification and matching in magnetic resonance images. In Richard A. Robb, editor, *Proc. SPIE 2359, Visualization in Biomedical Computing*, number September 1994, pages 160–169, sep 1994.

[100] J B Maintz and M A Viergever. A survey of medical image registration. *Medical image analysis*, 2(1):1–36, 1998.

[101] Julien Mairal, Francis Bach, and Jean Ponce. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(4):791–804, 2012.

[102] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Discriminative learned dictionaries for local image analysis. In *CVPR*, pages 1–8. IEEE, 2008.

[103] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Supervised dictionary learning. *arXiv preprint arXiv:0809.3083*, 2008.

[104] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *IJCV*, 43(1):7–27, 2001.

[105] Ravi Malladi, James A Sethian, and Baba C Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 17(2):158–175, 1995.

[106] Ravikanth Malladi, James a. Sethian, and Baba C. Vemuri. A topology-independent shape modeling scheme. *Geometric Methods in Computer Vision II*, 2031:246–258, 1993.

[107] G. Maso, J. M. Morel, and S. Solimini. A variational method in image segmentation: Existence and approximation results. *Acta Mathematica*, 168(1):89–151, 1992.

[108] T McInerney and D Terzopoulos. Topology adaptive deformable surfaces for medical image volume segmentation. *IEEE transactions on medical imaging*, 18(10):840–850, 1999.

[109] T McInerney and D Terzopoulos. T-snakes: topology adaptive snakes. *Medical image analysis*, 4(2):73–91, 2000.

[110] Marek Krzysztof Misztal and Jakob Andreas Bærentzen. Topology-adaptive interface tracking using the deformable simplicial complex. *ACM Transactions on Graphics*, 31(3):1–12, may 2012.

[111] Marek Krzysztof Misztal, Jakob Andreas Bærentzen, François Anton, and Kenny Erleben. Tetrahedral mesh improvement using multi-face retriangulation. *Proceedings of the 18th International Meshing Roundtable, IMR 2009*, pages 539–555, 2009.

[112] Marek Krzysztof Misztal, Robert Bridson, Kenny Erleben, Jakob Andreas Bærentzen, and François Anton. Optimization-based Fluid Simulation on Unstructured Meshes. In *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010)*, 2010.

[113] Marek Krzysztof Misztal, Kenny Erleben, Adam Bargteil, Jens Fursund, B Christensen, Jakob Andreas Bærentzen, and Robert Bridson. Multi-phase flow of immiscible fluids on unstructured moving meshes. In *Eurographics Symposium on Computer Animation*, pages 97–106. Eurographics Association, 2012.

[114] Marek Krzysztof Misztal, Kenny Erleben, Adam Bargteil, Jens Fursund, Brian Bunch Christensen, Jakob Andreas Bærentzen, and Robert Bridson. Multiphase flow of immiscible fluids on unstructured moving meshes. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):4–16, 2014.

[115] Scott A Mitchell and Stephen A Vavasis. Quality mesh generation in higher dimensions. *SIAM Journal on Computing*, 29(4):1334–1370, 2000.

[116] Matthias Müller. Fast and robust tracking of fluid surfaces. *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '09*, l:237, 2009.

[117] Matthias Müller, David Charypar, and Markus Gross. Particle-Based Fluid Simulation for Interactive Applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (5):154–159, 2003.

[118] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42:577–685, 1989.

[119] Tuan Nguyen, Vedrana Dahl, and Jacob Bærentzent. Multi-phase image segmentation with the adaptive deformable mesh. *Pattern Recognition Letters*, 2017.

[120] Tuan Nguyen, Vedrana Andersen Dahl, and J. Andreas Bærentzen. Template code for mesh cache: github.com/tuannt8/cache-template, 2016.

[121] Tuan T. Nguyen. Volume segmentation with tetrahedral mesh.

[122] Tuan T. Nguyen, Vedrana A. Dahl, and J. Andreas Bærentzen. Cachemesh, a Dynamics Data Structure for Performance Optimization. *Procedia Engineering*, 203:193–205, 2017.

[123] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168. IEEE, 2006.

[124] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[125] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[126] Stanley J. Osher. Fronts Propagating with Curvature Dependent Speed. *Computational Physics*, 79(1):1–5, 1988.

[127] NOBUYUKI Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[128] Nikhil R. Pal and Sankar K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[129] V N Parthasarathy, C M Graichen, and A F Hathaway. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design*, 15(3):255–261, 1994.

[130] Gabriel Peyré. Sparse modeling of textures. *Journal of Mathematical Imaging and Vision*, 34(1):17–31, 2009.

[131] Dzung L Pham, Chenyang Xu, and Jerry L Prince. Current methods in medical image segmentation. *Annual review of biomedical engineering*, 2(1):315–337, 2000.

[132] J P Pons and J D Boissonnat. Delaunay Deformable Models: Topology-Adaptive Meshes Based on the Restricted Delaunay Triangulation. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 13, pages 384–394, 2007.

[133] Jean Philippe Pons and Jean Daniel Boissonnat. Delaunay Deformable Models: Topology-Adaptive Meshes Based on the Restricted Delaunay Triangulation. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 13:384–394, 2007.

[134] Shaoping Quan, Jing Lou, and David P. Schmidt. Modeling merging and breakup in the moving mesh interface tracking method for multiphase flow simulations. *Journal of Computational Physics*, 228(7):2660–2675, 2009.

[135] Shaoping Quan and David P. Schmidt. A moving mesh interface tracking method for 3D incompressible two-phase flows. *Journal of Computational Physics*, 221:761–780, 2007.

[136] William T Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics (TOG)*, 2(2):91–108, 1983.

[137] Mikaël Rousson, Thomas Brox, and Rachid Deriche. Active unsupervised texture segmentation on a diffusion based feature space. In *CVPR*, volume 2, pages II–699. IEEE, 2003.

[138] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[139] Prasanna K Sahoo, SAKC Soltani, and Andrew K C Wong. A survey of thresholding techniques. *Computer vision, graphics, and image processing*, 41(2):233–260, 1988.

[140] Christophe Samson, Laure Blanc-Féraud, Gilles Aubert, Josiane Zerubia, Christophe Samson, Laure Blanc-Féraud, Gilles Aubert, Josiane Zerubia, and Multiphase Evolution. Multiphase Evolution and Variational Image Classification. Technical Report RR-3662, INRIA, apr 1999.

[141] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Transactions on Graphics*, 27(5):1, 2008.

[142] Jakob Santner, Markus Unger, Thomas Pock, Christian Leistner, Amir Saffari, and Horst Bischof. Interactive texture segmentation using random forests and total variation. In *BMVC*, pages 1–12. Citeseer, 2009.

[143] Guillermo Sapiro and Allen Tannenbaum. Affine invariant scale-space. *International journal of computer vision*, 11(1):25–44, 1993.

[144] Shankar P. Sastry, Emre Kultursay, Suzanne M. Shontz, and Mahmut T. Kandemir. Improved cache utilization and preconditioner efficiency through use of a space-filling curve mesh element- and vertex-reordering technique. *Engineering with Computers*, 30(4):535–547, 2014.

[145] Joachim Schöberl. NETGEN An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and visualization in science*, 1(1):41–52, 1997.

[146] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit*. Kitware, 4 edition, 2006.

[147] E. Seegyoung Seol and Mark S. Shephard. Efficient distributed mesh data structure for parallel automated adaptive analysis. *Engineering with Computers*, 22(3-4):197–213, 2006.

[148] J. A. Sethian. Curvature and the evolution of fronts. *Communications in Mathematical Physics*, 101(4):487–499, dec 1985.

[149] J A Sethian. A review of recent numerical algorithms for hypersurfaces moving with curvature dependent speed. *J. Differential Geometry*, 31:131–161, 1989.

[150] J. Shah. A common framework for curve evolution, segmentation and anisotropic diffusion. In *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, number I, pages 136–142. IEEE, 1996.

[151] M S Shephard. Meshing Environment for geometry based analysis. *International Journal of Numerical Methods in Engineering*, 47(1-3):169–190, 2000.

[152] Jr Shewchuk. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. *Unpublished manuscript*, pages 1–11, 2002.

[153] Karl Skretting and John Håkon Husøy. Texture classification using sparse frame-based representations. *EURASIP journal on applied signal processing*, 2006:102–102, 2006.

[154] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[155] Lucian Stãnculescu, Raphaëlle Chaine, and Marie Paule Cani. Freestyle: Sculpting meshes with self-adaptive topology. *Computers and Graphics (Pergamon)*, 35(3):614–622, 2011.

[156] Demetri Terzopoulos. On Matching Deformable Models to Images. *Topical Meeting on Machine Vision*, 12:160–167, 1986.

[157] Demetri Terzopoulos and Kurt Fleischer. *Deformable models*. 1988.

[158] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Constraints on deformable models: Recovering 3D shape and nonrigid motion, 1988.

[159] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Constraints on deformable models: Recovering 3D shape and nonrigid motion. *Artificial intelligence*, 36(1):91–123, 1988.

[160] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[161] Andy Tsai, Anthony Yezzi, and A.S. Willsky. Curve evolution implementation of the Mumford-Shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Transactions on Image Processing*, 10(8):1169–1186, 2001.

[162] Manik Varma and Rahul Garg. Locally invariant fractal features for statistical texture classification. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[163] Luminita A. Vese and Tony F. Chan. A multiphase level-set framework for image segmentation using the Mumford and Shah model. *International Journal of Computer Vision*, 50(3)(3):271–279, 2002.

[164] Jeffrey Scott Vitter. External Memory Algorithms and Data Structures. *ACM Computing Surveys (CSUR)*, 33(2):209–271, 2001.

[165] John F Wendt. *Computational fluid dynamics: an introduction*. Springer Science & Business Media, 2008.

[166] Bernard Widrow. The Rubber-Mask Technique-I, Pattern Measurement and Analysis. In *Learning Systems and Intelligent Robots*, pages 365–400. Springer US, Boston, MA, 1974.

[167] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Transactions on Graphics*, 28(3):1, jul 2009.

[168] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Transactions on Graphics*, 29(4):1, 2010.

[169] Yeu Wu. Chan Vese Active Contours without edges.

[170] Peter Wust, Johanna Gellermann, Jürgen Beier, Susan Wegner, Wolfgang Tilly, Jens Tröger, Detlev Stalling, H Oswald, H C Hege, P Deuflhard, and Others. Evaluation of segmentation algorithms for generation of patient models in radiofrequency hyperthermia. *Physics in Medicine & Biology*, 43(11):3295, 1998.

[171] Chenyang Xu and Jerry L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998.

[172] Chenyang Xu, Jerry L Prince, and Dzung L. Pham. Image Segmentation Using Deformable Models. In *Handbook of Medical Imaging*, pages 129–174. 2004.

[173] Chenyang Xu, Anthony Yezzi Jr, and Jerry L Prince. On the relationship between parametric and geometric active contours. In *The Asilomar Conference on Signals, Systems,and Computers*, volume 1, pages 483–489. IEEE, 2000.

[174] Anthony Yezzi Jr, Andy Tsai, and Alan Willsky. A statistical approach to snakes for bimodal and trimodal imagery. In *ICCV*, volume 2, pages 898–903. IEEE, 1999.

[175] Sung Eui Yoon and Peter Lindstrom. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1213–1220, 2006.

[176] Sung-Eui Yoon, Peter Lindstrom, Valerio Pascucci, and Dinesh Manocha. Cache-oblivious mesh layouts. *ACM Transactions on Graphics*, 24(3):886, 2005.

[177] Sung-Eui Eui Yoon and Dinesh Manocha. Cache-Efficient Layouts of Bounding Volume Hierarchies. *Computer Graphics Forum*, 25(3):507–516, sep 2006.

[178] Jihun Yu and Greg Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics*, 32(1):1–12, 2013.

[179] Jihun Yu, Chris Wojtan, Greg Turk, and Chee Yap. Explicit mesh surfaces for particle based fluids. *Computer Graphics Forum*, 31(2):815–824, 2012.

[180] A Zaharescu, E Boyer, and R Horaud. Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing and Multiview Reconstruction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1, 2010.

[181] Andrei Zaharescu, Edmond Boyer, and Radu Horaud. Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multiview Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):823–837, apr 2011.

[182] Y. Zhang, M. Brady, and S. Smith. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging*, 20(1):45–57, 2001.

[183] Hong-Kai Zhao, T. Chan, B. Merriman, and S. Osher. A Variational Level Set Approach to Multiphase Motion. *Journal of Computational Physics*, 127:179–195, 1996.